Teoría de Números

Bach. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería

Divisibilidad

- Dados dos enteros $\boldsymbol{a}$ y $\boldsymbol{b}$ se dice que $\boldsymbol{a}$ divide a $\boldsymbol{b}$ si existe un entero $\boldsymbol{c}$ tal que $\boldsymbol{b} = \boldsymbol{ac}$.
- Se denota de la siguiente manera:

$$\boldsymbol{a}|\boldsymbol{b}$$

Divisibilidad

Proposición 1

Sean $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d}, \boldsymbol{m}$ y $\boldsymbol{n}$ números enteros, si $\mathbf{d}|\boldsymbol{a}$ y $\mathbf{d}|\mathbf{b}$ entonces $\mathbf{d}|(\boldsymbol{ma} + \boldsymbol{nb})$

Números Primos

- Un número entero positivo $\mathbf{n}$ es denominado primo, si $\boldsymbol{n} > \mathbf{1}$ y sus únicos divisores son $\mathbf{1}$ y $\boldsymbol{n}$.
- Un número entero positivo diferente a $\mathbf{1}$ y que no es primo, se denomina compuesto.

Test de Primalidad

Podemos saber si un número es primo fácilmente en $\boldsymbol{O(n)}$, iterando por todos los posibles divisores.

¿ podemos reducir la complejidad?

Test de Primalidad

Teorema 1

Si $\mathbf{n}$ es un número compuesto, entonces $\mathbf{n}$ tiene al menos un divisor que es mayor que $\mathbf{1}$ y menor o igual a $\sqrt{\boldsymbol{n}}$.

Test de Primalidad

Demostración

Sea $n = ab$; donde a, b son enteros y $1 < a \leq b < n$, entonces:

$a \leq \sqrt{n}$, ya que si no caeríamos en una contradicción, porque tendríamos que $a, b > \sqrt{n}$ y por ende $ab > n$.

Test de Primalidad

```
bool isPrime( int n ){
    if( n<=1 ) return 0;
    for( int i=2; i*i<=n; ++i ){
        if( n%i == 0 ) return 0;
    }
    return 1;
}
```

Complejidad: $O(\sqrt{n})$

Problemas

UVA 382 - Perfection

UVA 10879 – Code Refactoring

UVA 294 – Divisors

UVA 1246 - Find Terrorists

Números Primos

¿Listar todos los números primos desde el 1 hasta n?

Si utilizamos el algoritmo explicado anteriormente por cada número del $\mathbf{1}$ al $\mathbf{n}$, tendríamos una complejidad de $\boldsymbol{O(\ n\sqrt{n}\ )}$.

¿ se puede reducir la complejidad?

Criba de Eratóstenes

Algoritmo que nos permite hallar todos los números primos desde el $\mathbf{1}$ hasta $\mathbf{n}$ de una manera eficiente.

Criba de Eratóstenes

1. Construimos un arreglo de tamaño $\mathbf{n}$, el cual nos indicará si un número es primo.

2. Inicialmente consideramos a todos los números como primos a excepción del $\mathbf{1}$.

3. Iteramos en orden creciente por los números empezando desde el $\mathbf{2}$.

   En cada iteración verificamos si el número en proceso no se encuentra marcado (primo), de ser así inmediatamente marcamos todos sus múltiplos como compuestos.

Criba de Eratóstenes

2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11
~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21
~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~	31
~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~	41
~~42~~	43	~~44~~	45	~~46~~	47	~~48~~	49	~~50~~	51

Criba de Eratóstenes

2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~	31
~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~	41
~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~	~~51~~

Criba de Eratóstenes

Continuamos así hasta haber recorrido todos los números.

2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~	31
~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~	41
~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~	~~51~~

Criba de Eratóstenes

Con este algoritmo obtenemos una complejidad de $O(n \log n)$

```
void sieve(){
    memset( isPrime, 1 , sizeof( isPrime) );
    isPrime[ 1 ] = 0;
    for( int i=2; i<=n; ++i ){
        if( isPrime[ i ] ){
            for( int j=i*2; j<=n; j+=i ) isPrime[ j ] = 0;
        }
    }
}
```

¿podemos optimizarlo más?

Criba de Eratóstenes

Usando el **Teorema 1**, solo necesitamos recorrer los números hasta la $\sqrt{\boldsymbol{n}}$, ya que todos los números compuestos deben ser tachados por alguno de éstos números.

Criba de Eratóstenes

```
void sieve(){
    memset( isPrime, 1 , sizeof( isPrime) );
    isPrime[ 1 ] = 0;
    for( int i=2; i*i<=n; ++i ){
        if( isPrime[ i ] ){
            for( int j=i*2; j<=n; j+=i ) isPrime[ j ] = 0;
        }
    }
}
```

Criba de Eratóstenes

Con una pequeña optimización más , nos queda un algoritmo en **O(n)**.

```
void sieve(){
    memset( isPrime, 1 , sizeof( isPrime) );
    isPrime[ 1 ] = 0;
    for( int i=2; i*i<=n; ++i ){
        if( isPrime[ i ] ){
            for( int j=i*i; j<=n; j+=i ) isPrime[ j ] = 0;
        }
    }
}
```

Cantidad de Números Primos

Sabemos que existen infinitos números primos, pero podemos estimar cuántos son menores a un número $\boldsymbol{x}$.

Se define la función $\boldsymbol{\pi}(\boldsymbol{x})$, siendo $\boldsymbol{x}$ un número real positivo, denotando el número de primos que no exceden a $\boldsymbol{x}$.

$\pi(4) = 2$

$\pi(5) = 3$

$\pi(10) = 4$

Cantidad de Números Primos

Teorema de los Números Primos

Cuando $\boldsymbol{x}$ es un número grande $\boldsymbol{x/lnx}$ es una buena aproximación de $\boldsymbol{\pi(x)}$.

x	$\pi(x)$	$x/\log x$	$\pi(x)/\dfrac{x}{\log x}$
10^3	168	144.8	1.160
10^4	1229	1085.7	1.132
10^5	9592	8685.9	1.104
10^6	78498	72382.4	1.085
10^7	664579	620420.7	1.071
10^8	5761455	5428681.0	1.061
10^9	50847534	48254942.4	1.054
10^{10}	455052512	434294481.9	1.048
10^{11}	4118054813	3948131663.7	1.043
10^{12}	37607912018	36191206825.3	1.039
10^{13}	346065535898	334072678387.1	1.036

Cantidad de Números Primos

Proposición 2

Dado un entero positivo n, es posible tener n números compuestos consecutivos.

Cantidad de Números Primos

Demostración

Consideremos los siguientes $\boldsymbol{n}$ enteros consecutivos:

$$(n+1)!+2, (n+1)!+3, \dots, (n+1)!+n+1$$

Para todo $2 \leq d \leq n+1$ sabemos que $d|(n+1)!$

Dado que $d|d$, usando la Proposición 1 entonces $d \mid (n+1)! + \mathrm{d}$

Distancia entre Primos Consecutivos

Se define la $\boldsymbol{n}$-ésima distancia (gap) entre números primos consecutivos como $\boldsymbol{g_n}$ igual a la diferencia entre el $(\boldsymbol{n}+\mathbf{1})$-ésimo y el $\boldsymbol{n}$-ésimo primo.

$$\boldsymbol{g_n = p_{n+1} - p_n}$$

$$g_1 = 1$$
$$g_3 = 2$$
$$g_4 = 4$$
$$g_9 = 6$$

Distancia entre Primos Consecutivos

- De la Proposición 2 podemos deducir que la distancia entre dos primos consecutivos puede llegar a ser muy grande.

- Para los valores que se manejan en los concursos esta distancia no es mucha.

Primo	Máximo gap
$p_n \leq 10^9$	282
$p_n \leq 10^{12}$	540
$p_n \leq 10^{18}$	1442

Problemas

Codeforces 230B - T-primes

UVA 543 – Goldbach’s Conjeture

Teorema Fundamental de la Aritmética

Todo número entero $\mathbf{n} > \mathbf{1}$ puede ser representado de forma única como producto de potencias de números primos.

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}$$

donde $p_1 < p_2 < \ldots < p_k$ son primos y α_i son enteros positivos.

Descomposición Factores Primos

Podemos descomponer un número $\mathbf{n}$ en $\mathbf{O}(\sqrt{n})$

```
void factorize( int n ){
    for( int i=2; i*i<=n; ++i ){
        if( n% i == 0 ){
            int exp = 0;
            while( n%i == 0 ){
                exp++;
                n/=i;
            }
            cout<< i <<"^"<<exp<<endl;
        }
    }
    if( n > 1 ){ // si n es primo
        cout<< n <<"^"<<1<<endl;
    }
}
```

Descomposición Factores Primos

- En la Criba además de saber si un número es primo , podemos guardar un divisor primo de cada número.
- Teniendo un factor primo de cada número, podemos descomponer cualquier otro de manera recursiva.
- Luego del costo de la criba $\boldsymbol{O}(\mathbf{n})$, podemos factorizar cualquier número $\mathbf{n}$ en complejidad $\boldsymbol{O}(\boldsymbol{log}\ \mathbf{n})$.

muy útil cuando tenemos muchas consultas

Descomposición Factores Primos

```
void extSieve(){
    memset( fact, 0, sizeof( fact ) );
    for( int i=2; i*i<=n; i++ ){
        if( !fact[ i ] ){
            for( int j= i*i ; j<=n; j+=i ) fact[ j ] = i;
        }
    }
    for( int i=2; i<=n; ++i ) if( !fact[ i ] ) fact[ i ] = i;
}

void factorize(int n){
    while( n > 1 ){
        int f = fact[ n ], exp = 0;
        while( n % f == 0 ) n/=f, exp++;
        cout<<f<<"^"<<exp<<endl;
    }
}
```

Problemas

Codeforces 757B – Bash’s Big Day

Codeforces 385C - Bear and Prime Numbers

Máximo Común Divisor

El máximo común divisor de dos enteros $\boldsymbol{a}$ y $\boldsymbol{b}$ (con al menos uno distinto a cero) es el más grande entero que divide a ambos.

Se denota de varias formas:

$$\mathbf{gcd}(\boldsymbol{a},\boldsymbol{b})\,,\mathbf{mcd}(\boldsymbol{a},\boldsymbol{b}),(\boldsymbol{a},\boldsymbol{b})$$

Máximo Común Divisor

Propiedades:

Sean a, b, k números enteros:

- $\gcd(a, 0) = a$
- $\gcd(a, ka) = a$
- $\gcd(a + \mathrm{kb}, b) = \gcd(a, \mathrm{b})$

Máximo Común Divisor

Demostración

$$\mathbf{gcd}(\boldsymbol{a} + \boldsymbol{k}\mathbf{b}, \boldsymbol{b}) = \boldsymbol{gcd}(\boldsymbol{a}, \boldsymbol{b})$$

- Todos los divisores comunes de $\boldsymbol{a}$ y $\boldsymbol{b}$ también son divisores de $\boldsymbol{a} + \boldsymbol{k}\mathbf{b}$ y $\boldsymbol{b}$
- Todos los divisores comunes de $\boldsymbol{a} + \boldsymbol{k}\mathbf{b}$ y $\boldsymbol{b}$ también son divisores $\boldsymbol{a}$ y $\boldsymbol{b}$
- Por lo tanto ambos tienen los mismos divisores comunes, por ello también el gcd.

Máximo Común Divisor

Teorema 3

Para todos los enteros $\boldsymbol{a}$ y $\boldsymbol{b}$ mayores o iguales a **0** (con al menos uno distinto a **0**).

$$\mathbf{gcd}(\boldsymbol{a}, \boldsymbol{b}) = \mathbf{gcd}(\boldsymbol{b}, \boldsymbol{a}\ \boldsymbol{mod}\ \boldsymbol{b})$$

Máximo Común Divisor

Demostración

- Podemos expresar a como $a = \mathrm{qb} + \mathrm{r}$, donde q, r son enteros y $0 \leq \mathrm{r} < \mathrm{b}$
- Entonces $\mathrm{r} = a \bmod b$
- Sabemos que: $\gcd(a, \mathrm{b}) = \gcd(a + \mathrm{kb}, b)$

$$\gcd(a, \mathrm{b}) = \gcd(a - \mathrm{qb}, b)$$

$$\gcd(a, \mathrm{b}) = \gcd(\mathrm{r}, \mathrm{b})$$

$$\gcd(a, \mathrm{b}) = \gcd(b, a \bmod b)$$

Algoritmo de Euclides

Podemos hallar el máximo común de dos números aplicando el Teorema 3 varias veces.

$Sea\ \boldsymbol{a} \geq \boldsymbol{b}$

$$\mathbf{gcd}(\boldsymbol{a}, \boldsymbol{b}) = \mathbf{gcd}(\boldsymbol{b}, \boldsymbol{r}_{\mathbf{1}}) = \mathbf{gcd}(\boldsymbol{r}_{\mathbf{1}}, \boldsymbol{r}_{\mathbf{2}}) = \cdots = \mathbf{gcd}(\boldsymbol{r}_{\mathbf{n-1}}, \mathbf{0}) = \boldsymbol{r}_{\mathbf{n-1}}$$

$$Donde, \qquad 0 < r_{n-1} < \cdots < r_2 < r_1 < b$$

En el caso que $\boldsymbol{b} < \boldsymbol{a}$, luego de una iteración llegamos al caso anterior.

Algoritmo de Euclides

Este algoritmo tiene complejidad $\boldsymbol{O}(\log \boldsymbol{b})$, donde $\boldsymbol{b}$ es el menor de los números.

```
int gcd( int a, int b ){
    while( b > 0 ){
        int aux = a;
        a = b;
        b = aux % b;
    }
    return a;
}
```

Primos Relativos

Dos enteros $\boldsymbol{a}$ y $\boldsymbol{b}$ son llamados primos relativos (coprimos) si $\boldsymbol{a}$ y $\boldsymbol{b}$ tienen el máximo común divisor igual a 1.

Problemas

Codeforces 664A - Complicated GCD

CodeChef – Cutting Recipes

Codeforces 75C – Modified GCD

Referencias

- Rosen, K. Elementary number theory and its applications.
- Topocoder https://goo.gl/nKOtvL

¡ Good luck and have fun !