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 1246 - Find Terrorists

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

x	$\pi(x)$	$x/\log x$	$\pi(x)/\frac{x}{\log x}$
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 n enteros consecutivos:

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

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 n-ésima distancia (gap) entre números primos consecutivos como g_n igual a la diferencia entre el $(n+1)$-ésimo y el n-ésimo primo.

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

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

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

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

```
const Long MX = 4e6;
bool isPrime[MX];

void sieve(Long n){
    fill(isPrime, isPrime + n + 1 , true);
    isPrime[0] = isPrime[1] = false;
    for(Long i = 2; i <= n; i++){
        for(Long j = 2 * i; j <= n; j += i){
            isPrime[j] = false;
        }

    }
}
```

¿Complejidad?

Criba de Eratóstenes - Complejidad

```
const Long MX = 4e6;
bool isPrime[MX];

void sieve(Long n){
    fill(isPrime, isPrime + n + 1 , true);
    isPrime[0] = isPrime[1] = false;
    for(Long i = 2; i <= n; i++){
        for(Long j = 2 * i; j <= n; j += i){
            isPrime[j] = false;
        }


    }
}
```

Analicemos el 2do for.

Por ejemplo, el número 2 tendrá que recorrer al 4 , 6 , 8 … es decir **todos los pares** entre 1 y n (a excepción del 2).

¿Cuántos pares hay entre 1 y n ?

$$\lfloor \frac{n}{2} \rfloor$$

De la misma forma, el 3 recorrerá a todos sus múltiplos ($\lfloor \frac{n}{3} \rfloor$) y luego el 4 y así sucesivamente.

Criba de Eratóstenes - Complejidad

Denotemos al número de operaciones totales del segundo for como T(n)

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \cdots$$

$$T(n) \leq \sum_{p=2}^{n} \frac{n}{p} = n \sum_{p=2}^{n} \frac{1}{p}$$ **(Serie Armónica)**

Criba de Eratóstenes - Complejidad

Analicemos la serie armónica. Denotemos a la serie armónica como H_n . Asumiendo que n + 1 es una potencia de 2 , tendríamos lo siguiente.

$$
\begin{aligned}
H_n &= \underbrace{1} + \underbrace{\frac{1}{2} + \frac{1}{3}} + \underbrace{\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}} + \underbrace{\frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{15}} + \cdots + \underbrace{\frac{1}{(n+1)/2} + \cdots + \frac{1}{n-1} + \frac{1}{n}} \\
&\leq \underbrace{1} + \underbrace{\frac{1}{2} + \frac{1}{2}} + \underbrace{\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}} + \underbrace{\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \cdots + \frac{1}{8}} + \cdots + \underbrace{\frac{1}{(n+1)/2} + \cdots + \frac{1}{(n+1)/2} + \frac{1}{(n+1)/2}} \\
&= \underbrace{\underbrace{1} + \underbrace{1} + \underbrace{1} + \underbrace{1} + \cdots + \underbrace{1}}_{k \text{ times}} \\
&= k \;=\; \log_2(n+1)
\end{aligned}
$$

Criba de Eratóstenes - Complejidad

De lo anterior, tenemos una cota para T(n)

$$T(n) \leq n \sum_{p=1}^{n} \frac{1}{p} = n \log_2(n+1)$$

$$\boxed{T(n) = O(n \log n)}$$

¿Podemos mejorarlo?

Criba de Eratóstenes

Utilizando la **Proposición 1**, el 2do for solo es necesario ejecutarlo cuando el i es primo

```
const Long MX = 4e6;
bool isPrime[MX];

void sieve(Long n){
    fill(isPrime, isPrime + n + 1 , true);
    isPrime[0] = isPrime[1] = false;
    for(Long i = 2; i <= n; i++){
        if(isPrime[i]){
            for(Long j = 2 * i; j <= n; j += i){
                isPrime[j] = false;
            }
        }
    }
}
```

¿Habrá mejorado la complejidad?

Criba de Eratóstenes - Complejidad

Denotemos al número de operaciones totales del segundo for como T(n)

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor + \left\lfloor \frac{n}{5} \right\rfloor + \cdots$$

$$T(n) = \sum_{\substack{p\ primo \\ p \leq n}} \frac{n}{p} = n \sum_{\substack{p\ primo \\ p \leq n}} \frac{1}{p}$$

Criba de Eratóstenes - Complejidad

Usaremos **El Teorema de los números primos**.

- ❑ La cantidad de números primos menores o iguales a *"n"* es aproximadamente $\frac{n}{\ln(n)}$
- ❑ Del teorema anterior, se deduce que el n-ésimo primo es aproximadamente $n \ln(n)$

Entonces podemos reemplazar eso en nuestra fórmula, trabajando el caso del primer primo (el 2) por separado para que el denominador no sea 0.

$$T(n) = n \sum_{\substack{p\ primo \\ p \leq n}} \frac{1}{p} \approx \frac{n}{2} + n \sum_{k=2}^{\frac{n}{\ln(n)}} \frac{1}{k \ln(k)}$$

Criba de Eratóstenes - Complejidad

Como la función $\frac{1}{k\ln(k)}$ es decreciente, podemos aplicar las sumas de Riemman para poder aproximar la suma con una **integral**

$$\sum_{k=2}^{\frac{n}{\ln(n)}} \frac{1}{k\ln(k)} \approx \int_{2}^{\frac{n}{\ln(n)}} \frac{dk}{k\ln(k)} = [\ln(\ln(k))]_{2}^{\frac{n}{\ln(n)}} = \ln(\ln \mathrm{n} - \ n) - \ln\ln 2 \approx \ln\ln n$$

Finalmente, reemplazando esto en nuestra expresión final, nos queda en notación Big O

$$\boxed{T(n) = O(n\log\log n) + O(n) = O(n\log\log n)}$$

Criba de Eratóstenes

Es posible realizar ciertas optimizaciones que no mejorarán la complejidad, pero reducirán un poco la constante.

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

Criba de Eratóstenes

```
const Long MX = 4e6;
bool isPrime[MX];

void sieve(Long n){
    fill(isPrime, isPrime + n + 1 , true);
    isPrime[0] = isPrime[1] = false;
    for(Long i = 2; i * i <= n; i++){
        if(isPrime[i]){
            for(Long j = 2 * i; j <= n; j += i){
                isPrime[j] = false;
            }
        }
    }
}
```

Criba de Eratóstenes

Podemos hacer otra optimización en el 2do for, al darnos cuenta que no es necesario empezar desde 2 * i para un número mayor a 2, ya que el 2 ya lo habría tachado. Tampoco empezar desde 3 * i para un número mayor a 3. Siguiendo esta lógica, deberíamos empezar en i * i.

```
const Long MX = 4e6;
bool isPrime[MX];

void sieve(Long n){
    fill(isPrime, isPrime + n + 1 , true);
    isPrime[0] = isPrime[1] = false;
    for(Long i = 2; i * i <= n; i++){
        if(isPrime[i]){
            for(Long j = i * i; j <= n; j += i){
                isPrime[j] = false;
            }
        }
    }
}
```

Criba Lineal

Aún con las anteriores optimizaciones, la complejidad se mantiene en O(n log log n).

Sin embargo, existe una versión óptima O(n). Para más información, revisar estas fuentes :

- CP - Algorithm: https://cp-algorithms.com/algebra/prime-sieve-linear.html
- Blog Codeforces - Sección Linear Sieve: https://codeforces.com/blog/entry/54090

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 $\mathbf{O}(\sqrt{n})$

```
struct Factor{
    Long base, exp;
    Factor(){}
    Factor(Long base, Long exp) : base(base), exp(exp){}
};

vector<Factor> factorize(Long n){
    vector<Factor> v;
    Long i = 2;
    while(i * i <= n){
        if(n % i == 0){
            Long exp = 0;
            while(n % i == 0){
                n /= i;
                exp++;
            }
            v.push_back(Factor(i , exp));
        }
        i++;
    }

    if(n > 1){ // n es primo
        v.push_back(Factor(n , 1));
    }

    return v;
}
```

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 $O(\mathbf{n})$, podemos factorizar cualquier número $\mathbf{n}$ en complejidad $O(log\ \mathbf{n})$.

muy útil cuando tenemos muchas consultas

Descomposición Factores Primos

```cpp
const Long MX = 2e7;
Long fact[MX];

void extSieve(Long n){
    for(Long i = 2; i < MX; i++){
        if(fact[i] == 0) {
            fact[i] = i; //primo
            for(Long j = i * i; j <= n; j += i){
                fact[j] = i;
            }

        }
    }
}
```

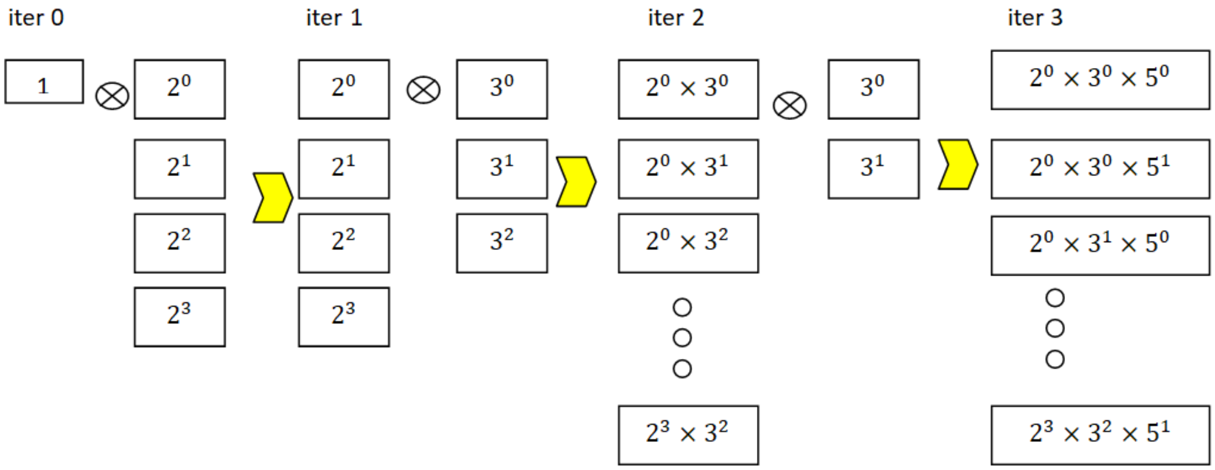
```cpp
struct Factor{
    Long base, exp;
    Factor(){}
    Factor(Long base, Long exp) : base(base), exp(exp){}
};

vector<Factor> factorize(Long x){ //O(log x)
    vector<Factor> v;
    while(x > 1){
        Long f = fact[x];
        Long exp = 0;
        while(x % f == 0){
            x /= f;
            exp++;
        }
        v.push_back(Factor(f ,exp));
    }
    return v;
}
```

Obtener todos los divisores de un número

La forma más simple es a través de bruteforce hasta la raíz, lo cuál nos deja complejidad O(sqrt(n)). Pero podemos hacerlo mejor, en O(# divisores) suponiendo que ya tenemos la descomposición canónica del número. Iremos procesando factor primo por factor primo. Por ejemplo :

$$n = 2^3 \times 3^2 \times 5^1$$



Obtener todos los divisores de un número

```
vector<Long> getDivisors(Long x) {
    vector<Long> ans = {1};
    while(x > 1) {
        Long f = fact[x];
        Long num = 1;
        Long sz = ans.size();
        while(x % f == 0) {
            num *= f;
            x /= f;
            for(Long i = 0; i < sz; i++) {
                ans.push_back(num * ans[i]);
            }
        }
    }
    //sort(ans.begin(), ans.end());
    return ans;
}
```

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

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

- 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

Podemos hallar el gcd de dos números aplicando el teorema 3 varias veces

$$Sea\ a \geq b.$$

$$Definamos\ r_0 = a\,, \qquad r_1 = b$$

$$\gcd(a\,, b\,) = \gcd(r_0\,, r_1) = \gcd(r_1\,, r_2) = \cdots = \gcd(r_{n-1}, 0) = \ r_{n-1}$$

$$Donde\ 0 = r_n < r_{n-1} < \cdots < r_2 < r_1 = b \leq r_0 = a$$

En caso a < b , luego de una iteración, llegaremos al caso anterior

Algoritmo de Euclides

```
Long gcd(Long a, Long b){
    if(b == 0){
        return a;
    }
    return gcd(b , a % b);
}
```

Un ajuste es necesario para manejar números negativos

```
Long gcd(Long a, Long b){
    if(b == 0) {
        return abs(a);
    }
    return gcd(b , a % b);
}
```

Algoritmo de Euclides - Complejidad

$$\gcd(a\ ,b\ ) = \gcd(r_0\ ,r_1) = \gcd(r_1\ ,r_2) = \cdots = \gcd\ (r_{n-2}\ ,r_{n-1}) = \gcd(r_{n-1},0)\ =\ r_{n-1}$$

$$Donde\ 0 = r_n < r_{n-1} < \cdots < r_2 < r_1\ = b \le r_0\ = a$$

$$Sabemos\ que\ \ r_0 = a\ ,\ \ r_1 = b,\ \ \ r_2 = a\ mod\ b = r_0\ mod\ r_1\ ,\ \ \ r_3 = r_1\ mod\ r_2\ , \ldots$$

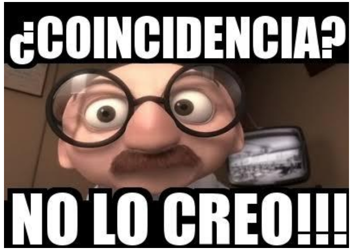
$$Podemos\ notar\ que\ r_i\ = r_{i-2}\ mod\ r_{i-1}\ \ \ ,para\ i \ge 2$$

$$Es\ decir\ \ r_{i-2} = q_{i-1}\ \times r_{i-1} + r_i\ \ \ldots(*)$$

$$Como\ todos\ los\ residuos\ son\ positivos, sabemos\ que\ r_i \ge 1\ ,\forall i$$

$$Además\ como\ r_i > r_{i-1}, entonces\ q_i \ge 1\ ,\forall\ i$$

Algoritmo de Euclides - Complejidad



$Utilizando\ el\ hecho\ de\ que\ r_{n-1} < r_{n-2}\ , y\ que\ r_{n-1} \geq 1\ , entonces\ r_{n-2} \geq 2$

$$De\ (*):\ r_{n-3} = q_{n-2} \times r_{n-2} + r_{n-1}$$

$$\Rightarrow r_{n-3} \geq 1 \times 2 + 1 = 2 + 1 = 3$$

$$De\ (*):\ r_{n-4} = q_{n-3} \times r_{n-3} + r_{n-2}$$

$$\Rightarrow r_{n-4} \geq 1 \times 3 + 2 = 3 + 2 = 5$$

$$De\ (*):\ r_{n-5} = q_{n-4} \times r_{n-4} + r_{n-3}$$

$$\Rightarrow r_{n-5} \geq 1 \times 5 + 3 = 5 + 3 = 8$$

$$De\ (*):\ r_i = q_{i+1} \times r_{i+1} + r_{i+2}$$

$$\Rightarrow r_i \geq r_{i+1} + r_{i+2}$$

$$r_n \geq 1 = f_2$$

$$r_{n-1} \geq 2 = f_3$$

$$r_{n-2} \geq 3 = f_4$$

$$r_{n-3} \geq 5 = f_5$$

$$r_{n-4} \geq 8 = f_6$$

Algoritmo de Euclides - Complejidad

Se puede demostrar por inducción que las cotas de los residuos siguen la secuencia de fibonacci, pero parece bastante obvio a simple vista.

De lo anterior podemos deducri que $b = r_1 \geq f_{n+1} (el\, n + 1\, ésimo\, fibonacci)$

$a = r_0 \geq f_{n+2} (el\, n + 2\, ésimo\, fibonacci)$

Por lo que $b \geq \frac{\varphi^{n+1} - (1-\phi)^{n+1}}{\sqrt{5}}$ $n = O(\log b)$

Finalmente la complejidad del algoritmo de euclides es $O(\log b)$

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.
- CP- Algorithm : https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html
- Topocoder https://goo.gl/nKOtvL

¡ Good luck and have fun !