Temas de Matemática I

Por Lautaro Lasorsa | Curso OIA UNLaM 2020

Divisor Común Mayor (GCD) | 1

Dados 2 enteros a y b definimos al $gcd(a,b)$ (el Divisor Común Mayor entre a y b) al mayor entero d tal que $d|a$ y $d|b$ (d divide a a y d divide a b).

El Algoritmo de Euclides permite obtener el $gcd(a,b)$ en $O(log(max(a,b)))$

Divisor Común Mayor (GCD) | 2

La siguiente función implementa el Algoritmo de Euclides.

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

https://pastebin.com/G9Q91cLC

Divisor Común Mayor (GCD) | 3

El *gcd* tiene las siguientes propiedades:

- Conmutativa: $gcd(a,b) = gcd(b,a)$
- Asociativa: $gcd(a,gcd(b,c)) = gcd(gcd(a,b),c)$ (esto permite hacer consultas en rango usando Segment Tree)
- Idem Potente: $gcd(a,a) = a$
- El 1 es elemento absorbente: $gcd(a,1) = 1$
- El 0 es elemento neutro: $gcd(a,0) = a$

Múltiplo Común Menor (LCM) | 1

Sean dos enteros a, b, el Múltiplo Común Menor entre ellos es el menor entero $c = lcm(a,b)$ tal que $a|c$ y $b|c$ (c es múltiplo de a y múltiplo de b).

Propiedad: $gcd(a,b) * lcm(a,b) = a*b \Leftrightarrow lcm(a,b) = (a*b)/gcd(a,b)$

Esto indica una forma fácil de calcular el $lcm(a,b)$ siendo que ya sabemos calcular el $gcd(a,b)$.

Múltiplo Común Menor (LCM) | 2

El *lcm* tiene las siguientes propiedades:

- Conmutativa: $lcm(a,b) = lcm(b,a)$
- Asociativa: $lcm(a,lcm(b,c)) = lcm(lcm(a,b),c)$ (esto permite hacer consultas en rango usando Segment Tree)
- Idem Potente: $lcm(a,a) = a$
- El 1 es elemento neutro: $lcm(a,1) = a$
- El 0 es elemento absorbente: $lcm(a,0) = 0$

Potencia Binaria | 1

El objetivo es calcular eficientemente a^N (N un número natural) en complejidad $O(log(N))$, cuando la forma usual haciendo N productos tiene complejidad $O(N)$.

La forma de hacerlo eficientemente es la siguiente recursión:

- Si N es 0: La respuesta es 1.
- Si N es par: Calculo $a^{N/2}$, y luego lo elevo al cuadrado. Importante: El exponente se reduce a la mitad en una sola operación.
- Si N es impar: Calculo a^{N-1} y luego lo multiplico por a.

Potencia Binaria | 2

La siguiente función implementa dicha recursión de manera eficiente.

```
long long BinPow(long long a, long long n){
    if(n == 0) return 1;
    if(n%2==1) return BinPow(a,n-1)*a%mod;
    long long a2 = BinPow(a,n/2);
    return a2*a2%mod;
}
```

https://pastebin.com/bLaCDgnV

Potencia Binaria | 3

Importante: Notar que dada cualquier operación asociativa *Op* se puede utilizar el mismo algoritmo, para calcular *Op(a,Op(a,...)) N* veces en *O(log(N)*O(Op))*.

Es simplemente reemplazar donde se utiliza el producto * por la operación en cuestión.

Dos ejemplos notables son el producto de matrices y la composición de funciones.

Implementación de Fracciones | 1

Sea un número racional $a = p/q$ con p y q coprimos ($gcd(p,q) = 1$), una forma de representarlo que permite no perder precisión a lo largo de las cuentas es guardar explícitamente los enteros p y q coprimos.

Una forma sencilla de comparar las fracciones a/b vs c/d es comparar $a*d$ vs $c*b$. **Para este método de comparación no es necesario que gcd(a,b) = 1 o que gcd(c,d) = 1.**

Implementación de Fracciones | 2

La siguiente es una posible implementación.

```
typedef long long intl;
struct fraccion{
    intl p,q;
};
bool operator<(const fraccion & a, const fraccion & b){
    return a.p*b.q < b.p*a.q;
}
```

https://pastebin.com/muyapqgx

GRACIAS POR VER