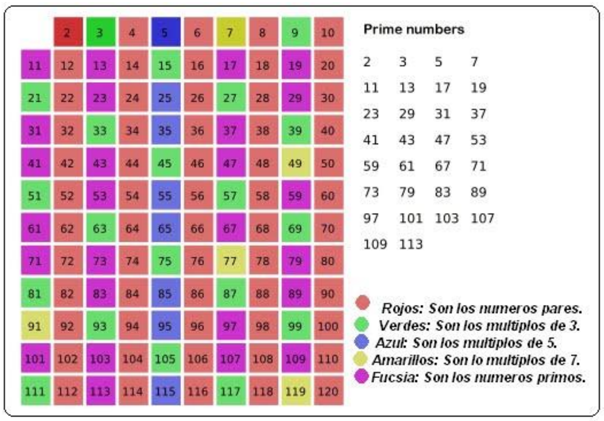
Algoritmos Matemáticas

Generar Primos:

Para generar primos de manera eficiente, podemos recurrir a la criba de Eratóstenes. Este algoritmo solo es útil cuando la cantidad y el tamaño de los primos no es demasiado grande.

Se forma una tabla con todos los números naturales comprendidos entre 2 y n, y se van tachando los números que no son primos de la siguiente manera: Comenzando por el 2, se tachan todos sus múltiplos; comenzando de nuevo, cuando se encuentra un número entero que no ha sido tachado, ese número es declarado primo, y se procede a tachar todos sus múltiplos, así sucesivamente. El proceso termina cuando el cuadrado del mayor número confirmado como primo es mayor que n.



$$O(n \ \log\log n)$$

Código:

```
#include <stdio.h>
#include <math.h>
#define MAXN 1000
using namespace std;
int n = 100;
bool primos[MAXN+2];
// Primos se inicializa con 0
// Genera todos los primos menores que 100
void criba(){
    int lim = sqrt(n);
    primos[2] = true;
    for( int i = 3; i < n; primos[i] = true,i+=2);
    for( int i = 3; i < lim; i+=2)
        if( primos[i] )
            for( int k = i*i; k <= n; k+=i)
                primos[k] = false;
}
```

Exponenciación Binaria Modular

A veces es necesario elevar un número a una potencia en específico, sin embargo, cuando la potencia es bastante grande, es necesario reducir los cálculos. Cuando calculamos a^b, solemos iterar desde 1 hasta b, sin embargo esto se vuelve bastante ineficiente cuando b es muy grande. La exponenciación binaria se basa en la siguiente función recursiva:

$$a^b = \begin{cases} 1, & b = 0 \\ \left(a^{\frac{b}{2}}\right)^2, & en\ otro\ caso \end{cases}$$

$$Complejidad: O(\log b)$$

Lo que se hace es guardar el resultado de $a^{\frac{b}{2}}$ para no tener que volver a calcularlo, y después simplemente elevar al cuadrado. Con esto podemos mejorar considerablemente nuestro algoritmo reduciendo la complejidad de O(b) a O(logb).

Código:

```
#include <stdio.h>
#include <math.h>
#define Mod 10007
using namespace std;
// calcula a^b modulo Mod
// Si eliminamos %Mod calcula solamente a^b
long long int potencia(long long int a, long long int b){
    if( b == 0 ) return 1;
    if( b == 1 ) return a%Mod;
    long long int mitad = potencia(a,b/2)%Mod;
    mitad = (mitad*mitad)%Mod;
    if( b%2 == 0 ) return mitad;
    return (mitad * a)%Mod;
}
```

Máximo común divisor y Mínimo común múltiplo

Para calcular el Máximo común divisor utilizaremos la siguiente función recursiva:

$$Gcd(a, b) = Gcd(b, a\ mod\ b)\ y\ se\ sabe\ que\ \mathrm{Gcd}(0, \mathrm{a}) = \mathrm{a}$$
$$Además\ Gcd(a, b)Lcm(a, b) = ab$$

Código:

```
#include <stdio.h>
using namespace std;
int gcd( int a, int b){
    if( b == 0 ) return a;
    return gcd(b,a%b);
}
int lcm( int a, int b){
    return a/gcd(a,b)*b;
}
```

Obtener Inverso Modular

En programación competitiva es usual que algún resultado te lo pidan modulo algún número. Los problemas vienen cuando los números son muy grandes y tienes que hacer divisiones modulo un número. Un ejemplo son las combinaciones modulo ***p primo***

$$\binom{n}{k} mod\ p = \left(\frac{n!}{k!\,(n-k)!}\right) mod\ p \neq \left(\frac{n!\ mod\ p}{(k!\,(n-k)!)\ mod\ p}\right)$$

Es *incorrecto* asumir que para sacar la división modulo p se tiene que dividir por el dividendo modulo p.

Supongamos que nosotros queremos dividir modulo ***p primo***, para esto necesitamos encontrar el *inverso modular.*

$$Por\ el\ pequeño\ teorema\ de\ fermat:$$

$$a^{p-1} \equiv 1\ mod\ p \ \rightarrow\ a^{p-1}a^{-1} \equiv\ a^{-1}\ mod\ p\ \rightarrow a^{p-2}\ \equiv a^{-1}\ mod\ p$$

Por lo tanto, si nosotros queremos **dividir** por **a** modulo ***p***, entonces tenemos que **multiplicar** por $a^{p-2}\ mod\ p$

Código:

```
#include <stdio.h>
#include <math.h>
#define Mod 10007
using namespace std;
long long int potencia(long long int a, long long int b){
    if( b == 0 ) return 1;
    if( b == 1 ) return a%Mod;
    long long int mitad = potencia(a,b/2)%Mod;
    mitad = (mitad*mitad)%Mod;
    if( b%2 == 0 ) return mitad;
    return (mitad * a)%Mod;
}
long long int inversoModular(long long int a){
    return potencia(a,Mod-2);
}
```