Matemática

Números prímos

Descrição

- sieve: Crivo de Eristótenes, armazena no vetor primes os primos até **n** e no bitset bs se o o numero é ou não primo. em complexidade **O(n log log n)**;
- is_prime_sieve: Calcula um numero primo caso **sievesize** seja maior ou igual que √N em complexidade **O(√n/log n)**
- is_prime: Calcula se um número é ou não primo de maneira rápida sem depender do crivo em complexidade **O(√n)**

```
using ll = long long;
const long long MAX = 10000009;
ll sievesize;
bitset<MAX> bs;
vector<ll> primes;

void sieve(ll n){
    sievesize = n+1;
    bs.set();
    bs[0]=bs[1]=0;
    for(ll i=2; i<=sievesize; ++i){
        if(bs[i]){
            for(ll j=i*i; j<=sievesize; j+=i)
                bs[j]=0;
            primes.push_back(i);
        }
    }
}
bool is_prime_sieve(ll n){
    if(n<=(ll)sievesize) return bs[n];
    for(size_t i=0; i<primes.size() and primes[i]*primes[i]<=n; ++i)
        if(n%primes[i] == 0) return false;
    return true;
}
bool is_prime(ll n){
    if(n<0) n=-n;
    if(n<5 or n%2==0 or n%3==0)
        return (n==2 or n==3);
    ll maxP = sqrt(n)+2;
    for(ll p=5; p<maxP; p+=6){
        if( p<n and n%p==0 ) return false;
        if( p+2<n and n%(p+2)==0 ) return false;
    }
    return true;
}
```

Divisores

Descrição

- gdc: Calcula maior divisor comum em complexidade **O(log a+b)**
- lmc: Calcula o menor múltiplo comum em complexidade **O(log a+b)**
- num_div: Calcula o números de divisores de N em complexidade **O(√n)**
- num_div: Calcula o número de divisóres de um número mais rapdamente com os primos pré-calculados até √n, complexidade **O(n^(1/pi))**

```
using ll = long long;

ll gdc(ll a, ll b){
    ll rest;
    do{
        rest = a%b;
        a=b;
        b=rest;
    }while(rest!=0);
    return a;
}
ll lmc(ll a, ll b){
    return a/gdc(a, b)*b;
}
ll num_div(ll n){
    ll ans=0;
    for(ll i=1; i*i<=n; ++i)
        if(n%i == 0)
            ans+=( i==n/i ? 1 : 2);
    return ans;
}
ll num_div2(ll n){
    ll i=0, p=primes[i], ans=1;
    while(p*p<=n){
        ll power = 0;
        while(n%p==0) n/=p, ++power;
        ans *= (power+1);
        p = primes[++i];
    }
    if(n!=1) ans*=2;
    return ans;
}
```