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;
}
```

Aritimética modular

Descrição

- gdc: Calcula maior divisor comum em complexidade **O(log a + log b)**
- lmc: Calcula o menor múltiplo comum em complexidade **O(log a + log b)**
- ext_gcd: Algorítimo estendido de euclides complexidade:**O(log a + log b)**
- num_div: Calcula o números de divisores de N em complexidade **O(√n)**
- mod_inv: Calcula inverso modular de a em módulo m complexidade:**O(log a+b)**
- mod_exp: Calcula a elevado à b em módulo m em complexidade:**O(log b)**
- mod_mul: Calcula (a*b)%m sem overflow em complexidade:**O(log a + log b)**
- diophantine: Acha uma solução na equação no formato ax+by=c, sendo x e y as incognitas; Após a divisão de a, b e c pelo mdc(a, b) as outras soluções são x=x0+bt, y=y0-at; complexidade:**O(log a + log b)**
- preprocess_fat: Pré-processa os fatorias em módulo m até MAXN em complexidade **O(MAXN)**

```
using ll = long long;
template<typename T>
T gcd(T a, T b){
    return b==0 ? a : gcd(b, a%b);
}
template<typename T>
T lmc(T a, T b){
    return a*(b/gcd(a, b));
}
template<typename T>
T ext_gcd(T a, T b, T& x, T& y){
    if(b==0){
        x=1; y=0; return a;
    }
    else{
        T g = ext_gcd(b, a%b, y, x);
        y-=a/b*x; return g;
    }
}
template<typename T>
T num_div(T n){
    T ans=0;
    for(T i=1; i*i<=n; ++i)
        if(n%i == 0)
            ans+=( i==n/i ? 1 : 2);
    return ans;
}
template<typename T>
T mod_inv(T a, T m){
    T x, y;
    ext_gcd(a, m, x, y);
    return (x%m+m)%m;
}
```

```
template<typename T>
T mod_mul(T a, T b, T m){
    T x=0, y=a%m;
    while(b>0){
        if(b%2==1) x=(x+y)%m;
        y=(y*2)%m; b/=2;
    }
    return x%m;
}
template<typename T>
T mod_exp(T a, T b, T m){
    if(b == 0) return (T) 1;
    T c = mod_exp(a, b/2, m);
    c = (c*c)%m;
    if(b%2 != 0) c=(c*a)%m;
    return c;
}
template<typename T>
void diophantine(T a, T b, T c, T& x, T& y){
    T d = ext_gcd(a, b, x, y);
    x *= c/d; y*= c/d;
}
#define MAXN 1000009
using ll = long long;
ll fat[MAXN];
void preprocess_fat(ll m){
    fat[0] = 1;
    for(ll i=1; i<MAXN; ++i)
        fat[i]=(i*fat[i-1])%m;
}
```