欧拉筛

```
bool not_prime[MAXN];
int prime[MAXN];
int top = 0;

not_prime[0] = not_prime[1] = true;
for (int i = 2; i * i < MAXN; ++i)
{
    if (!not_prime[i]) prime[top++] = i;
    for (int j = 0; j < top && i * prime[j] < MAXN; ++j)
    {
        not_prime[i * prime[j]] = true;
        if (i % prime[j] == 0) break;
    }
}
```

$O(n)$

gcd -> Greatest Common Divisor -> 最大公约数
lcm -> Least Common Multiple -> 最小公倍数

欧几里得算法 (辗转相除法)

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}

int lcm(int a, int b)
{
    return a * b / gcd(a, b);
}

<algorithm> 头文件中自带__gcd函数
```

拓展欧几里得算法

$ax + by = \gcd(a, b)$

由欧几里得算法原理, 得到 $bx + (a - a/b * b)y = \gcd(a, b)$

即 $ay + b * (x - (a/b) * y) = \gcd(a, b)$

```
void extended_gcd(int a, int b, int &d, int &x, int &y)
{
    if (!b) { d = a; x = 1; y = 0; }
    else
    {
        extended_gcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}
```

```
const int MOD = 1000000007;
int mod_inverse[MAXN];
mod_inverse[0] = 0;
mod_inverse[1] = 1;

for (int i = 2; i < MAXN; ++i)
{
    mod_inverse[i] = (MOD - MOD / i)
                     * mod_inverse[MOD % i] % MOD;
}
```

$O(n)$

```
int CRT(const int n, const int a[], const int r[])
{
    int m2, r2, i, d, x, y, c, t;
    int m1 = a[0];
    int r1 = r[0];
    bool no_solution = false;
    for (i = 1; i < n; ++i)
    {
        m2 = a[i];
        r2 = r[i];
        extended_gcd(m1, m2, d, x, y);
        c = r2 - r1;
        if (c % d)
        {
            no_solution = true;
            break;
        }
        t = m2 / d;
        x = (c / d * x % t + t) % t;
        r1 += m1 * x;
        m1 = m1 / d * m2;
    }
    if (no_solution) return -1;
    if (n == 1 && r1 == 0) return m1;
    return r1;
}
```

```
m2 = a[i];
r2 = r[i];
extend_gcd(m1, m2, d, x, y);
//d=extend_gcd(m1,m2)
//x*m1+y*m2=d
c = r2 - r1;
if (c % d)//对于方程m1*x+m2*y=c, 如果c不是d的倍数就无整数解
{
    no_solution = true;
    break;
}
t = m2 / d;
//对于方程m1x+m2y=c=r2-r1,若(x0,y0)是一组整数解,那么
//(x0+k*m2/d,y0-k*m1/d)也是一组整数解(k为任意整数)
//其中x0=x*c/d,y0=y*c/d
x = (c / d * x % t + t) % t;
//保证x0是正数, 因为x+k*t是解, (x%t+t)%t也必定是正数解
//(必定存在某个k使得(x%t+t)%t=x+k*t)
r1 += m1 * x;
//新求的r1就是前i组的解, Mi=m1*x+M(i-1)=r2-m2*y
//(m1为前i个m的最小公倍数)
//对m2取余时, 余数为r2
//对以前的m取余时, Mi%m=m1*x%m+M(i-1)%m=M(i-1)%m=r
m1 = m1 / d * m2;
```