质数

> n 范围以内质数个数大约 $\frac{x}{lnx}$ 个

试除法判断质数

```
bool is_prime(int n)
{
    if(n == 1)   return false;
    for(int i = 2; i <= n / i; i ++)
        if(n % i == 0)
            return false;
    return true;
}
```

筛质数

```
void get_primes(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])    primes[cnt ++] = i;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0)   break;
        }
    }
}
```

分解质因数

朴素质因数分解

```
vector<PII> divide(int n)
{
    vector<PII> res;
    for(int i = 2; i <= n / i; i ++)
        if(n % i == 0)
        {
            int num = 0;
            while(n % i == 0)
                n /= i, num ++;
            res.push_back({i, num});
        }
    if(n >= 2)   res.push_back({n, 1});

    return res;
}
```

最小质因数分解质因数

```
void init(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i]) primes[cnt ++] = i, minp[i] = i;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            minp[primes[j] * i] = primes[j];
            if(i % primes[j] == 0)
                break;
        }
    }
}

void divide(int n)
{
    while(n > 1)
    {
        int cnt = 0;
        while(val % minp[n] == 0)
            val /= minp[n], cnt ++;
    }
}
```

约数

> int范围约数个数最多1500，long long范围约数最多大约1e5个

试除法求约数

```
vector<int> get_divisors(int n)
{
    vector<int> ans;

    for(int i = 1; i <= n / i; i ++)
        if(n % i == 0)
        {
            ans.push_back(i);
            if(n / i != i)
                ans.push_back(n / i);
        }
    return ans;
}
```

最大公约数

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

欧拉函数

欧拉函数

```
int get_euler(int n)
{
    int res = n;
    for(int i = 2; i <= n / i; i ++)
        if(n % i == 0)
        {
            res = res / i * (i - 1);
            while(n % i == 0)
                n /= i;
        }
    if(n > 1)
        res = res / n * (n - 1);

    return res;
}
```

筛法求欧拉函数

```
void get_eulers(int n)
{
    phi[1] = 1;
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])
            primes[cnt ++] = i, phi[i] = i - 1;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0)
            {
                phi[primes[j] * i] = phi[i] * primes[j];
                break;
            }
            phi[primes[j] * i] = phi[i] * (primes[j] - 1);
        }
    }
}
```

快速幂

```
int qmi(int a, int b, int k)
{
    int res = 1;
    while(b)
    {
        if(b & 1)   ans = (LL)ans * a % k;
        b >>= 1, a = (LL)a * a % k;
    }
    return res % k;
}
```

扩展欧几里得(ax + by = gcd(a, b))

```
LL ex_gcd(LL a, LL b, LL& x, LL& y)
{
    if(!b)   return x = 1, y = 0, a;

    LL d = ex_gcd(b, a % b, y, x);
    y -= x * (a / b);
    return d;
}
```

逆元

欧拉定理

如果(a, b) = 1, 那么$a^{\varphi(b)} \equiv 1$(mod b)

扩展欧拉定理

$$a^b \equiv \begin{cases} a^{b\%\varphi(p)} & (a,p)=1 \\ a^b & (a,p)\neq 1, b<\varphi(p) \\ a^{b\%\varphi(p)+\varphi(p)} & (a,p)\neq 1, b\geq\varphi(p) \end{cases} \qquad (mod p)$$

线性求逆元

```
void init(int n)
{
    inv[1] = 1;
    for(int i = 2; i <= n; i ++)
        inv[i] = (LL)(mod - mod / i) * inv[mod % i] % mod;
}
```

快速幂求逆元(mod为质数)

```
int get_inv(int a)
{
    return qmi(a, mod - 2, mod);
}
```

快速幂加欧拉公式求逆元(mod不为质数且a与mod互质)

```
int get_inv(int a)
{
    return qmi(a, get_euler(mod) - 1, mod);
}
```

扩展欧几里得求逆元

```
int get_inv(int a)
{
    int x, y;
    ex_gcd(a, mod, x, y);
    return x;
}
```

同余方程组(x = bi(%ai))

中国剩余定理(ai两两互质)

```
LL crt(int a[], int b[], int n)
{
    LL M = 1, res = 0;
    for(int i = 0; i < n; i ++)
        M *= a[i];

    for(int i = 0; i < n; i ++)
    {
        LL Mi = M / a[i], x, y;
        ex_gcd(Mi, a[i], x, y);     //t = Mi在ai下的逆元

        res += (LL)b[i] * Mi % M * x % M;
        res = (res % M + M) % M;
    }
    return res;
}
```

同余方程组

```
bool excrt(LL& a1, LL& b1, LL a2, LL b2)
{
    LL k1, k2, d = ex_gcd(a1, a2, k1, k2);

    if((b2 - b1) % d)   return false;

    LL t = a2 / d;
    k1 = ((i128)(b2 - b1) / d * k1 % t + t) % t;      //最小正整数解

    b1 += k1 * a1;
    a1 *= a2 / d;
```

```
    return true;
}
```

组合数

预处理组合数

```
void init(int n)
{
    for(int i = 0;i <= n;i++)
        for(int j = 0;j <= i;j++)
        {
            if(!j)    c[i][j] = 1;
            else    c[i][j] = (c[i - 1][j - 1] + c[i - 1][j]) % mod;
        }
}
```

预处理阶乘

```
void init(int n)
{
    fact[0] = infact[0] = inv[1] = 1;
    for(int i = 1; i <= n; i ++)
    {
        fact[i] = (LL)fact[i - 1] * i % mod;
        if(i > 1)    inv[i] = (LL)(mod - mod / i) * inv[mod % i] % mod;
        infact[i] = (LL)infact[i - 1] * inv[i] % mod;
    }
}
```

求组合

```
int C(int a, int b)
{
    int up = 1, down = 1;
    for(int i = a, j = 1; j <= b; j ++, i --)
        up = (LL)up * i % mod, down = (LL)down * j % mod;

    return (LL)up * qmi(down, mod - 2, mod) % mod;
}
```

卢卡斯定理

```
int lucas(LL a, LL b, int p)           //p为质数
{
    if (a < p && b < p) return C(a, b, p);
    return (LL)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}
```

卡特兰数

> ans = C(2 * n, n) - C(2 * n, n - 1) = $\frac{C(2*n,n)}{n+1}$

错排

```
void init(int n)
{
    f[0] = 1, f[1] = 0;
    for(int i = 2; i <= n; i ++)
        f[i] = (LL)(f[i - 1] + f[i - 2]) * (i - 1) % mod;
}
```

高斯消元

```
int gauss()  // 高斯消元，答案存于a[i][n]中，0 <= i < n
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ )  // 找绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i <= n; i ++ ) swap(a[t][i], a[r][i]);  // 将绝对值最大的行换到最顶端
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c];  // 将当前行的首位变成1
        for (int i = r + 1; i < n; i ++ )  // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1; // 有无穷多组解
    }

    for (int i = n - 1; i >= 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];

    return 0; // 有唯一解
}
```

莫比乌斯反演

莫比乌斯函数:

1.μ为莫比乌斯函数，定义为:

$$\mu(n)=\begin{cases}1 & n=1\\ 0 & \text{含有平方因子}\\ (-1)^k & k\text{为}n\text{的本质不同质因子个数}\end{cases}$$

2.莫比乌斯函数性质:

$\sum_{d|n}$μ(d) =

$$\begin{cases}1 & n=1\\ 0 & n\neq 1\end{cases}$$

莫比乌斯反演:

1.若 F(n) = $\sum_{d|n}$ f(d), 则 f(n) = $\sum_{d|n}$ μ(d) × F($\frac{n}{d}$)

2.若 F(n) = $\sum_{n|d}$ f(d), 则 f(n) = $\sum_{n|d}$ μ($\frac{d}{n}$) × F(d)

线性求莫比乌斯函数:

```
void init(int n)
{
    mu[1] = 1;

    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])
            primes[cnt ++] = i, mu[i] = -1;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0)
                break;
            mu[primes[j] * i] = -mu[i];
        }
    }
}
```

BSGS

bsgs

```
int bsgs(int a, int b, int p)
{
    if(1 % p == b % p)    return 0;            //特判0

    int k = sqrt(p) + 1;
    map<int, int> M;

    for(int i = 0, j = b % p; i < k; i ++)
        M[j] = i, j = (LL)j * a % p;

    int ak = qmi(a, k, p);
```

```
    for(int i = 1, j = ak; i <= k; i ++)
    {
        if(M.count(j))    return i * k - M[j];
        j = (LL)j * ak % p;
    }
    return -1;
}
```

ex_bsgs

```
int ex_bsgs(int a, int b, int p)
{
    b = (b % p + p) % p;

    if(1 % p == b % p)              //特判0
        return 0;

    int x, y, d = ex_gcd(a, p, x, y);
    if(d > 1)
    {
        if(b % d)
            return -INF;
        ex_gcd(a / d, p / d, x, y);
        return ex_bsgs(a, (LL)b / d * x % (p / d), p / d) + 1;
    }
    return bsgs(a, b, p);
}
```

FFT

```
while((1 << bit) < n + m + 1) bit ++;
    tot = 1 << bit;

for(int i = 0; i < tot; i ++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));

void fft(Complex a[], int inv)
{
    for(int i = 0; i < tot; i ++)
        if(i < rev[i])
            swap(a[i], a[rev[i]]);

    for(int mid = 1; mid < tot; mid <<= 1)
    {
        Complex w1 = { cos(PI / mid), inv * sin(PI / mid) };
        for(int i = 0; i < tot; i += mid * 2)
        {
            Complex wk = { 1, 0 };
            for(int j = 0; j < mid; j ++, wk = wk * w1)
            {
                Complex x = a[i + j], y = wk * a[i + j + mid];
                a[i + j] = x + y, a[i + j + mid] = x - y;
```

```
                }
            }
        }
    }
```