快速幂

```
long long pow(long long a, long long b) {
    long long res = 1;
    while(b) {
        if(b & 1) res = res * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}
```

龟速乘

```
ll mul(ll a, ll b, ll mod)
{
    ll res = 0;
    while(b > 0)
    {
        if(b & 1) res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
```

```
    }
    return res;
}
```

高斯消元

```
double a[105][106];

bool gauss(int n){
    //枚举列
    for(int i = 1; i <= n; ++i){
        int id = i;
        //找到该列中的最大系数
        for(int j = i; j <= n; ++j){
            id = fabs(a[j][i]) > fabs(a[id][i]) ? j : id;
        }
        // 交换
        for(int j = 1; j <= n + 1; ++j){
            swap(a[i][j], a[id][j]);
        }
        // 系数为零说明没有唯一解
        if(!a[i][i]){
            return false;
        }
        // 消去对应列的其他元
        for(int j = 1; j <= n; ++j){
            if(j != i){
                double temp = a[j][i] / a[i][i];
                for(int k = i + 1; k <= n + 1; ++k){
                    a[j][k] -= a[i][k] * temp;
                }
            }
        }
    }

    return true;
}
```

埃氏筛素数

```
bool is_prime[N + 1]; // default true
is_prime[0] = is_prime[1] = false;
for(int i = 2; i <= N; ++i) {
    if(is_prime[i]) {
        for(int j = i << 1; j <= N; j += i) {
            is_prime[j] = false;
        }
```

```
    }
}
```

线性筛素数

```
vector<int>p;
vector<bool>prime(N, true);

void Euler_sieve(){
    for(int i = 2; i < N; ++i){
        if(prime[i])p.push_back(i);
        for(int j = 0; j < p.size() && i * p[j] < N; ++j){
            prime[i * p[j]] = false;
            if(i % p[j] == 0)break;
            // i % p[j] == 0
            // 换言之，i 之前被 p[j] 筛过了
            // 由于 pri 里面质数是从小到大的，所以 i 乘上其他的质数的结果一定会被
            // p[j] 的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break
            // 掉就好了
        }
    }
}
```

gcd/exgcd

扩展欧几里得用于解决$ax+by=\gcd(a,b)$的一组可行解

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

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

ll Exgcd(ll a, ll b, ll &x, ll &y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    ll d = Exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b);
    return d;
}
```

逆元

费马小定理

要求模数是质数。

- [x] = pow(x, MOD - 2);

扩欧

相当于求线性同余方程

$$
\begin{aligned}
\mathrm{inv} &\equiv \frac{1}{x} \pmod M \\
\mathrm{inv}x &\equiv 1 \pmod M \\
\mathrm{inv}x + kM &\equiv 1 \pmod M
\end{aligned}
$$

`std::tie(gcd, inv[x], _) = exgcd(x, MOD);`

若 gcd 不为 1 ，则逆元不存在。

线性求逆元

- [x] = (MOD - MOD / x) * inv[MOD % x] % MOD;

口诀：减除乘模

组合数

$$
\begin{aligned}
\binom{n}{m} &= \binom{n-1}{m-1} + \binom{n-1}{m} \\
&= \frac{n!}{m!(n-m)!}
\end{aligned}
$$

卢卡斯定理

要求：模数不大，是质数

$$\binom{n}{m} \bmod p = \binom{\lfloor \frac{n}{p} \rfloor}{\lfloor \frac{m}{p} \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

```
void init(){
    fac[0] = fac[1] = 1 ;
    for(int i = 2 ; i <= N ; ++i){
        fac[i] = fac[i - 1] * i % p ;
    }


    inv[0] = 1 ;
    //求n!的逆元
    inv[N] = qpow(f[N] , p - 2) ;          //费马小定理
    for(int i = N - 1 ; i > 0 ; --i){
        inv[i] = inv[i + 1] * (i + 1) % p ;
    }

}

long long C(long long n, long long m, long long p) {
    return fac[n] * inv[m] % p * inv[n - m] % p;
}

long long Lucas(long long n, long long m, long long p) {
  return m == 0 ? 1 : (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
}
```

皮克定理

Pick 定理：给定顶点均为整点的简单多边形，皮克定理说明了其面积 S 和内部格点数目 n 、边上格点数目 m 的关系：$S = n + \frac{m}{2} - 1$。

枚举

枚举子集

```
for(int i = 0; i < (1 << n); ++i){
    for(int j = i; j; j = (j - 1) & i){

    }
}
```

分解质因子

```
vector<int>d;   //d[x] = smallest divisor of x
vector<int>factor[N];       //factors[x] = prime factors of x, including
duplicates(重复)


d.assign(N, 1);

for(int i = N - 1; i > 1; --i){
    for(int j = i; j < N; j += i){
        d[j] = i;
    }
}
for(int i = 2; i < N; ++i){
    for(int j = i; j != 1; j /= d[j]){
        factor[i].push_back(d[j]);
    }
}
```