试除法判定质数 —— 模板题 AcWing 866. 试除法判定质数

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

试除法分解质因数 —— 模板题 AcWing 867. 分解质因数

```
void divide(int x)
{
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            int s = 0;
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}
```

朴素筛法求素数 —— 模板题 AcWing 868. 筛质数

```
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];         // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (st[i]) continue;
        primes[cnt ++ ] = i;
        for (int j = i + i; j <= n; j += i)
            st[j] = true;
    }
}
```

线性筛法求素数 —— 模板题 AcWing 868. 筛质数

```
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];         // st[x]存储x是否被筛掉

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;
        }
    }
}
```

试除法求所有约数 —— 模板题 AcWing 869. 试除法求约数

```
vector<int> get_divisors(int x)
{
    vector<int> res;
    for (int i = 1; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res.push_back(i);
            if (i != x / i) res.push_back(x / i);
        }
    sort(res.begin(), res.end());
    return res;
}
```

约数个数和约数之和 —— 模板题 AcWing 870. 约数个数, AcWing 871. 约数之和

如果 N = $p_1^{c_1} * p_2^{c_2} * ... *p_k^{c_k}$

约数个数：$(c_1 + 1) * (c_2 + 1) * ... * (c_k + 1)$

约数之和：$(p_1^0 + p_1^1 + ... + p_1^{c_1}) * ... * (p_k^0 + p_k^1 + ... + p_k^{c_k})$

欧几里得算法 —— 模板题 AcWing 872. 最大公约数

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

求欧拉函数 —— 模板题 AcWing 873. 欧拉函数

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

    return res;
}
```

筛法求欧拉函数 —— 模板题 AcWing 874. 筛法求欧拉函数

```
int primes[N], cnt;     // primes[]存储所有素数
int euler[N];           // 存储每个数的欧拉函数
bool st[N];         // st[x]存储x是否被筛掉

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

快速幂 —— 模板题 AcWing 875. 快速幂

求 m^k mod p，时间复杂度 O($logk$)。

```
int qmi(int m, int k, int p)
{
    int res = 1 % p, t = m;
    while (k)
    {
        if (k&1) res = res * t % p;
        t = t * t % p;
        k >>= 1;
    }
    return res;
}
```

扩展欧几里得算法 —— 模板题 AcWing 877. 扩展欧几里得算法

```
// 求x, y，使得ax + by = gcd(a, b)
int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1; y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a/b) * x;
    return d;
}
```

高斯消元 —— 模板题 AcWing 883. 高斯消元解线性方程组

```
// a[N][N]是增广矩阵
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ )   // 找到绝对值最大的行
            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
}
```

递推法求组合数 —— 模板题 AcWing 885. 求组合数 I

```
// c[a][b] 表示从a个苹果中选b个的方案数
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] + c[i - 1][j - 1]) % mod;
```

通过预处理逆元的方式求组合数 —— 模板题 AcWing 886. 求组合数 II

首先预处理出所有阶乘取模的余数fact[N]，以及所有阶乘取模的逆元infact[N]

如果取模的数是质数，可以用费马小定理求逆元

```
int qmi(int a, int k, int p)    // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}
```

```
// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i ++ )
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
```

Lucas定理 —— 模板题 AcWing 887. 求组合数 III

若p是质数，则对于任意整数$1 <= m <= n$，有：

C(n, m) = C(n % p, m % p) * C(n / p, m / p) (mod p)

```
int qmi(int a, int k, int p)  // 快速幂模板
{
    int res = 1 % p;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

int C(int a, int b, int p)  // 通过定理求组合数C(a, b)
{
    if (a < b) return 0;

    LL x = 1, y = 1;  // x是分子，y是分母
    for (int i = a, j = 1; j <= b; i --, j ++ )
    {
        x = (LL)x * i % p;
        y = (LL) y * j % p;
    }

    return x * (LL)qmi(y, p - 2, p) % p;
}

int lucas(LL a, LL b, int p)
{
    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;
}
```

分解质因数法求组合数 —— 模板题 AcWing 888. 求组合数 IV

当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：

1. 筛法求出范围内的所有质数
2. 通过$C(a,b) = a!/b!/(a-b)!$这个公式求出每个质因子的次数。
   $n!$中p的次数是$n/p + n/p^2 + n/p^3 + \ldots$
3. 用高精度乘法将所有质因子相乘

```
int primes[N], cnt;     // 存储所有质数
int sum[N];     // 存储每个质数的次数
bool st[N];     // 存储每个数是否已被筛掉

void get_primes(int n)      // 线性筛法求素数
{
    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;
        }
    }
}

int get(int n, int p)       // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b)       // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
        c.push_back(t % 10);
```

```
        t /= 10;
    }

    return c;

}

get_primes(a);  // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )     // 求每个质因数的次数
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )     // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);
```

卡特兰数 —— 模板题 AcWing 889. 满足条件的01序列

给定n个0和n个1，它们按照某种顺序排成长度为$2n$的序列，满足任意前缀中0的个数都不少于1的个数的序列的数量为：$Cat(n) = C(2n, n)/(n+1)$

NIM游戏 —— 模板题 AcWing 891. Nim游戏

给定N堆物品，第i堆物品有Ai个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。取走最后一件物品者获胜。两人都采取最优策略，问先手是否必胜。

我们把这种游戏称为NIM博弈。把游戏过程中面临的状态称为局面。整局游戏第一个行动的称为先手，第二个行动的称为后手。若在某一局面下无论采取何种行动，都会输掉游戏，则称该局面必败。

所谓采取最优策略是指，若在某一局面下存在某种行动，使得行动后对面面临必败局面，则优先采取该行动。同时，这样的局面被称为必胜。我们讨论的博弈问题一般都只考虑理想情况，即两人均无失误，都采取最优策略行动时游戏的结果。

NIM博弈不存在平局，只有先手必胜和先手必败两种情况。

定理： NIM博弈先手必胜，当且仅当 A1 ^ A2 ^ … ^ An != 0

公平组合游戏ICG

若一个游戏满足：

1. 由两名玩家交替行动；
2. 在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关；

3. 不能行动的玩家判负；

则称该游戏为一个公平组合游戏。

NIM博弈属于公平组合游戏，但城建的棋类游戏，比如围棋，就不是公平组合游戏。因为围棋交战双方分别只能落黑子和白子，胜负判定也比较复杂，不满足条件2和条件3。

有向图游戏

给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子。两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。该游戏被称为有向图游戏。

任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个节点，并且从每个局面向沿着合法行动能够到达的下一个局面连有向边。

Mex运算

设S表示一个非负整数集合。定义$mex(S)$为求出不属于集合S的最小非负整数的运算，即：

$mex(S) = min{x}, x$属于自然数，且x不属于S

SG函数

在有向图游戏中，对于每个节点x，设从x出发共有k条有向边，分别到达节点$y_1, y_2, \dots, y_k$，定义$SG(x)$为x的后继节点$y_1, y_2, \dots, y_k$的SG函数值构成的集合再执行$mex(S)$运算的结果，即：

$SG(x) = mex(SG(y_1), SG(y_2), \dots, SG(y_k))$

特别地，整个有向图游戏G的SG函数值被定义为有向图游戏起点s的SG函数值，即SG(G) = SG(s)。

有向图游戏的和 —— 模板题 AcWing 893. 集合-Nim游戏

设$G_1, G_2, \dots, G_m$是m个有向图游戏。定义有向图游戏G，它的行动规则是任选某个有向图游戏G_i，并在G_i上行动一步。G被称为有向图游戏$G_1, G_2, \dots, G_m$的和。

有向图游戏的和的SG函数值等于它包含的各个子游戏SG函数值的异或和，即：

SG(G) = SG(G_1) ^ SG(G_2) ^ ... ^ SG(G_m)

定理

有向图游戏的某个局面必胜，当且仅当该局面对应节点的SG函数值大于0。

有向图游戏的某个局面必败，当且仅当该局面对应节点的SG函数值等于0。

> 作者：yxc
>
> 链接：https://www.acwing.com/blog/content/406/
>
> 来源：AcWing

> 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。