数论初步

PS：以下部分定理没有证明，如果有读者想要了解定理的具体证明，请自行百度，本文限于篇幅~~(只是因为笔者自己不会)~~，对部分定理的证明不作讨论。

目录

本文讲啥

本文主要讲的是ACM中的数论基础内容~~（以后可能会再写一篇ACM的数论进阶内容）~~，侧重应用，证明都是瞎证的，严谨的证明请观众姥爷自行百度

线性筛筛素数

埃氏筛

埃氏筛用每个素数来筛掉它的倍数，剩下的就是素数，时间复杂度是$O(nloglogn)$

为啥能正确地筛？

每一个合数，都可以被质因数分解，且根据**唯一分解定理**，这个分解是唯一的，所以只要拿每个素数把它的倍数都筛掉，就能所有合数筛掉

板子

```
bool check[maxn];
std::vector<ll> prime;

for (ll i = 2; i <= n; ++i)
    if (!check[i]) {
        prime.push_back(i);
        for (ll j = i * i; j <= n; j += i)
            check[j] = 1;
    }
```

为啥j从i^2开始？

上面的板子中第二层循环就是用素数筛掉它的倍数的过程，j从$2i$开始循环，那为什么板子里是从i^2开始呢？实际上，这是一个小小的优化，因为在用素数i来筛它的倍数时，区间$[2, i-1]$中的所有素数已经将它的倍数都筛过了，所以$2i$，$3i$，$\ldots$，$(i-1)*i$，都可以跳过

欧氏筛[1]

埃氏筛已经巨快了，然而对于每个合数，都很可能被筛多次，如16会被2筛，也会被4筛，下面介绍的欧氏筛可以做到每个合数就只被筛一次

咋筛的？

欧氏筛能正确地筛主要基于以下两点:

- 任何一个合数都可以被表示为一个素数和一个数的乘积
- 假设合数$q = x * y$，且x也是合数，y是素数，那么同样合数$x = a * b$，且a是素数，则$q = x * y = a * b * y = a * (b * y)$，即$q$被另一个素数$a$，和另一个合数$b * y$表示，不论$a$和$y$谁打谁小，大的一个总能转到小的那个

第二点是理解板子里的第二层循环的$if(i \% prime[j] == 0)$的关键，每个合数只要被最小的能筛它的素数筛掉就行了，这样保证每个合数只会被筛一次

板子

```
bool check[maxn];
ll prime[maxn], tot = 0;

for(ll i = 2; i <= n; i++) {
    if(!check[i])
        prime[++tot] = i;
    for(int j = 1; j <= tot && i * prime[j] <= n;
j++) {
        check[i * prime[j]] = 1;
        if(i % prime[j] == 0)
            break;
    }
}
```

除了筛素数，还能干啥？

欧氏筛不仅能筛素数，还可以用来筛积性函数，如欧拉函数$\phi(n)$，莫比乌斯函数$\mu(n)$等，这些不在本文讨论范围内，感兴趣的读者可以自行百度

快速幂、快速~~(龟速)~~乘和矩阵快速幂都是用来干啥的？

快速幂可以以$O(logb)$的复杂度求出$a^b \bmod mod$，快速乘是用来防止乘法溢出的，矩阵快速幂可以快速求出$A^b \bmod mod$，其中A是一个矩阵，$A \bmod mod \iff \forall a\ in\ A$，$a\% = mod$，$a$，$b$是两个long long型整数

先讲一下模运算的一点性质

$$(a*b)\%p = (a\%p)*(b\%p)\%p$$
$$(a^b)\%p = ((a\%p)^b)\%p$$

基本思想

我们先来看下面的一个等式

$$b = (b_{63}b_{62}\cdots b_1b_0)_2$$

其中b_i是从低位开始数b的第i位二进制值，则

$$a^b = a^{(b_{63}b_{62}\cdots b_1b_0)_2} = a^{b_{63}*2^{63}} * a^{b_{62}*2^{62}} * \cdots * a^{b_0*2^0}$$

矩阵快速幂的话只要把乘法换成矩阵乘法即可

相比于快速幂，快速乘可能用的地方少一点，如果在乘法过程中模数比较大，如mod是$1e15$的数量级的，那么难以避免地会产生溢出，这时可以用快速乘把乘法转化为加法来避免溢出

$$a * b = a * (b_{63}b_{62} \cdots b_1 b_0)_2 = a * (b_{63} * 2^{63}) + a * (b_{62} * 2^{62}) + \cdots + a * (b_0 * 2^0)$$

板子

```
struct Mat{
    int m[N][N];
    inline void init() {
        for(int i = 0; i < N; i++)
            for(int j = 0; j < N; j++)
                m[i][j] = 0;
    }
}E; // E初始化为单位矩阵

// 快速幂
inline ll qpow(ll a, ll b, ll mod) {
    ll ans = 1L;
    while (b) {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}

// 龟速乘
inline ll qmul(ll a, ll b, ll mod) {
    ll ans = 0L;
    while(b) {
        if(b & 1) ans = (ans + a) % mod;
        a <<= 1;
        b >>= 1;
    }
}

inline Mat mul(Mat a, Mat b, ll mod) {
```

```
    Mat c;
    c.init();
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            for(int k = 0; k < N; k++)
                c.m[i][j] = (c.m[i][j] + a.m[i][k]
* b.m[k][j]) % mod;
    return c;
}

// 矩阵快速幂
inline Mat mpow(Mat a, ll b, ll mod) {
    Mat ans = E;
    while (b) {
        if (b & 1)
            ans = mul(ans, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return ans;
}
```

矩阵快速幂怎么用？

基本上不会有题目直接要求矩阵$A^b \bmod mod$，一般是通过$f(n)$与前几项的递推关系求$f(n)$的值，这个n一般贼大，比如$1e18$，~~不大的话用啥矩阵快速幂直接递推就完了~~，比如问斐波那契数列的第n项，$n <= 1e18$，这时由于$f(n) = f(n-1) + f(n-2)$，有

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{n-2} \begin{bmatrix} f(2) \\ f(1) \end{bmatrix}$$

我们记 $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{n-2}$，那么$f(n) \equiv A.m[0][0] * f(2) + A.m[0][1] * f(1) \pmod{mod}$，这种类型的题都可以像这样构造矩阵来求解

在？来道例题

- POJ 3233 **Matrix Power Series**

  思路：题目涉及到求矩阵的高次幂，可以往矩阵快速幂方向想，构造递推关系，从式子 $S_k = A + A^2 + \cdots + A^k$，可以看出$S_k = S_{k-1} * A + A$，那么就可以构造矩阵了

$$\begin{bmatrix} S_k \\ A \end{bmatrix} = \begin{bmatrix} A & E \\ 0 & E \end{bmatrix} \begin{bmatrix} S_{k-1} \\ A \end{bmatrix} = \begin{bmatrix} A & E \\ 0 & E \end{bmatrix}^{k-1} \begin{bmatrix} S_1 \\ A \end{bmatrix} = \begin{bmatrix} A & E \\ 0 & E \end{bmatrix}^{k-1} \begin{bmatrix} A \\ A \end{bmatrix}$$

数论分块[2]

干啥的？

数论分块常用来解决下面这种问题

$$\sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor$$

其中$\left\lfloor \frac{n}{i} \right\rfloor$表示$n$除$i$向下取整

当然暴力地$O(n)$肯定能求出来，但是对于很多题$O(n)$是不够的，其实只要我们列出前几项，就会发现$\left\lfloor \frac{n}{i} \right\rfloor$会有一段连续的相同值，如$9/5$，$9/6$，$9/7$，$9/8$都是$1$，所以只要用这一段的长度乘以这一段的值就可以快速求和

板子

```
ll ans = 0L;
for(ll l = 1L, r = 0L; l <= n; l = r + 1) {
    ll num = n / l;
    r = n / num;
    ans += (r - l + 1) * num;
}
```

然而一般不会出现裸题，数论分块一般和其他的内容结合起来，或者式子需要进行适当变换，如

$$\sum_{i=1}^{n}(k \bmod i) = \sum_{i=1}^{n}(k - \left\lfloor \frac{k}{i} \right\rfloor * i) = k * n - \sum_{i=1}^{n} \left\lfloor \frac{k}{i} \right\rfloor * i$$

这样可以用等差数列来求$\lfloor \frac{k}{i} \rfloor$相等的一段的值

欧拉定理

内容

若正整数a，n互质， 即$\gcd(a, n)$则

$$a^{\phi(n)} \equiv 1 \pmod n$$

其中，$\phi(n)$是欧拉函数，等于区间$[1, n-1]$上与n互质的数的个数

推论——欧拉降幂公式

若正整数a，n互质，那么对于任意正整数b，有

$$a^b \equiv a^{b \bmod \phi(n)} \pmod n$$

欧拉函数的性质：

- $\phi(p^n) = p^n - p^{n-1}$

- 若m，n互质，$\phi(mn) = \phi(m) * \phi(n)$，即欧拉函数是积性函数
- 设 $n = p_1^{a_1} * p_2^{a_2} * \cdots * p_k^{a_k}$，则 $\phi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \cdots * (1 - \frac{1}{p_k})$，且$n$的因子个数为$(a_1 + 1) * (a_2 + 1) * \cdots * (a_k + 1)$

咋算欧拉函数ϕ？

1. 欧氏筛筛欧拉函数，可以以$O(n)$的复杂度计算出phi[1]到phi[n]

```
phi[1] = 1;
check[1] = true;
for (int i = 2; i <= n; i++) {
    if (!check[i]) {
        prime[++tot] = i;
        phi[i] = i - 1;
    }
    for (int j = 1; j <= tot && i * prime[j]
<= n; j++) {
        check[i * prime[j]] = 1;
        if (i % prime[j])
            phi[i * prime[j]] = phi[i] *
(prime[j] - 1);
        else {
            phi[i * prime[j]] = phi[i] *
prime[j];
            break;
        }
    }
}
```

2. 单计算$\phi[n]$，$O(\sqrt{n})$

```
int euler(int x) {
    int res = x;
    for(int i = 2; i * i <= x; 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;
}
```

能干啥？

- 能降幂~~(不然叫欧拉降幂公式干啥)~~

例题

- HDU 4704 **SUM**

  思路：用高中排列组合可以看出$\sum_{i=1}^{N} S(i) = 2^{N-1}$，快读读入N，边读边取模，需要注意的是要对$MOD-1$取摸，然后快速幂就完事了

费马小定理

内容

对于素数p，任意整数a，均满足

$$a^p \equiv a \pmod p$$

如果a不能被能被p整除，即$p \nmid a$，则上式可化为

$$a^{p-1} \equiv 1 \pmod p$$

本质

费马小定理是欧拉定理的特例，因为对于质数p，$\phi(p)=p-1$

能干啥？

- 先讲讲乘法逆元是啥

> 若有$a*b\equiv 1 \pmod p$，则称b是a在模p意义下的乘法逆元，或a是b在模p意义下的乘法逆元，记为$inv(a)$或$inv(b)$

- 快速幂求乘法逆元，适用于模数p为质数的情况

$$a^{p-1}\equiv a^{p-2}*a\equiv 1 \pmod p$$

所以，a^{p-2}就是a在模p意义下的逆元，即$inv(a)=a^{p-2}$，快速幂$O(logn)$就能求

例题

- HDU 1576 **A/B**

  思路：因为$(A/B)\equiv A*inv(B)\equiv (A \bmod 9973)*inv(B) \pmod{9973}$，$A \bmod 9973=n$，9973是个质数，可以快速幂求$inv(B)$

威尔逊定理

内容

当p为素数时，

$$(p-1)!\equiv -1 \pmod p$$

且这时一个充要条件，即当式子成立时，p是素数

咋用？

俺也不知道，蒟蒻没做过这类的题~~(，也许可以用来化简一些式子~~

扩展欧几里得$^{[3]}$

内容

大家都知道欧几里得算法就是求$\gcd(a, b)$的辗转相除法

```
inline int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

那扩展欧几里得是个啥呢？在介绍扩欧之前，先了解一下**贝祖定理**或称**裴蜀定理**：

> 对于任意整数a, b，$ax + by = \gcd(a, b)$一定有整数解，且对于任何整数x, y，$ax + by$必然是$\gcd(a, b)$的倍数

贝祖定理还有一个重要的推论：

> 对于任意整数a, b，a, b互质 $\iff \exists x, y \in Z$，使得$ax + by = 1$

那如何求出$ax + by = \gcd(a, b)$的解呢？这时**扩展欧几里得**闪亮登场

我们可以注意到在辗转相除法达到递归边界时，即当$b == 0$时，$a = \gcd(a, b)$，我们得出此时的$ax + by = \gcd(a, b)$的解是$x = 1, y = 0$，再反过来推回去(有点像数学归纳法反过来)

假设我们要求$ax+by=\gcd(a,b)$的整数解，而我们已经通过递归求出了$b*x_1+(a\%b)*y_1=\gcd(b,a\%b)$，这时我们把$a\%b=a-\left\lfloor\frac{a}{b}\right\rfloor*b$带入前面的式子有

$$b*x_1+(a-\left\lfloor\frac{a}{b}\right\rfloor*b)*y_1=a*y_1+b*(x_1-\left\lfloor\frac{a}{b}\right\rfloor*y_1)$$

可以发现$x=y_1,y=x_1-\left\lfloor\frac{a}{b}\right\rfloor*y_1$，这样两个相邻的递归状态间的$x,y$可以转换，在求gcd的同时就能顺便解方程

板子

```
typedef long long ll;
inline ll exgcd(ll a, ll b, ll &x, ll &y) {
    if(!b) {
        x = 1L; y = 0L;
        return a;
    }
    ll gcd = exgcd(b, a % b, x, y);
    ll tmp = y;
    y = x - a / b * y;
    x = tmp;
    return gcd;
}
```

作用

- 很显然第一种用法就是求$ax+by=\gcd(a,b)$的整数解，需要注意的是求出来的只是一组解
- 求乘法逆元，大概算是求解线性同余方程的一种

  前文已经说过快速幂可以在模数p为素数的前提下，以$O(logn)$的复杂度求出n关于p的乘法逆元$inv(n)=x$，但是如果p不是素数，就不能用快速幂求逆元了，这时可以用扩欧来求乘法逆元

  考虑式子

  $$n*x\equiv 1\pmod p$$

可以转化为

$$n*x+p*y=1$$

两边同乘$\gcd(n,p)$得到

$$n*x*\gcd(n,p)+p*y*\gcd(n,p)=\gcd(n,p)$$

这时用扩欧可以求出$x*\gcd(n,p)$和$y*gcd(n,p)$，进而可以求出x

- 扩展中国剩余定理，限于篇幅这个内容不作讨论~~也许以后会填坑~~

例题

- CF 963A **Alternating Sum**

  思路：我们以字符串长度将和式进行分块，每len为一块，我们先求出第一块的和$\sum\limits_{i=0}^{len-1} s_i a^{n-i} b^i$并将它记为$sum$，那么后面一块的和就等与$sum*b^{len}/a^{len}$，总计$block=n/len$块，每块的和构成等比数列，总的和$S=sum*((b^{len}/a^{len})^{block}-1)/(b^{len}/a^{len}-1)$，

  通过扩欧求出a在模$p=1e9+9$下的乘法逆元$inv(a)$，那么在模p意义下，$b^{len}/a^{len}\equiv b^{len}*x^{len} \pmod p$，我们记$q=b^{len}*x^{len}\%p$，$S\equiv(sum\%p)*(q^{block}-1)/(q-1) \pmod p$，那我们再求一次在模$p$意义下$q-1$的乘法逆元$inv(q-1)$，即可求解。需要注意的是如果$q==1$，$inv(q-1)$就没有意义，所以需要特判，如果$q==1$，$S=sum*block$

中国剩余定理

内容

设正整数$m_1,m_2,...,m_k$两两互质，则同余方程组

$$\begin{cases} x \equiv a_1 \pmod {m_1} \\ x \equiv a_2 \pmod {m_2} \\ \vdots \\ x \equiv a_k \pmod {m_k} \end{cases}$$

有整数解，且在模$M = \prod_{i=1}^{k} m_i$下的解是唯一的，解为

$$x \equiv \sum_{i=1}^{k} a_i * M_i * M_i^{-1} \pmod M$$

其中$M_i = M/m_i$，M_i^{-1}为M_i在模m_i下的乘法逆元

瞎**证明

由定义$\gcd(m_i, M_i) = 1$，设

$$x^* = \sum_{i=1}^{k} a_i * M_i * M_i^{-1}$$

由于$a_i * M_i * M_i^{-1} \equiv a_i \pmod {m_i}$，且对$\forall j \neq i, a_i * M_i * M_i^{-1} \equiv 0 \pmod {m_j}$，所以

$$x^* = a_i * M_i * M_i^{-1} + \sum_{j \neq i} a_j * M_j * M_j^{-1} \equiv a_i + \sum_{j \neq i} 0 \equiv a_i \pmod {m_i}$$

故x^*是同余方程组的一个解。

设x_1和x_2都是同余方程组的解，那么对于$\forall i \in [1, k], x_1 - x_2 \equiv 0 \pmod {m_i}$，这说明$x_1 - x_2$是$m_i (\forall i \in [1, k])$的倍数，而$m_1, m_2, \ldots, m_k$两两互质，这说明$x_1 - x_2$是$M$的倍数，所以同余方程组在模$M = \prod_{i=1}^{k} m_i$下的解是唯一的，解为

$$x \equiv \sum_{i=1}^{k} a_i * M_i * M_i^{-1} \pmod M$$

作用

- 显然，中国剩余定理是用来求解模数互质的线性同余方程组的

扩展

既然中国剩余定理是求解模数互质的线性同余方程组的，好学的读者们肯定会问，那要是模数不互质呢？关于模数不互质的线性同余方程组，我们暂且不进行讨论~~以后大概会填坑~~

裸题

- TJOI 2009/洛谷P3868 **猜数字**

  思路：从题意看出需要求解方程组$n \equiv a_i \pmod{b_i}, i \in [1, k]$，是中国剩余定理的裸题

卢卡斯定理

内容

对于正整数n, m，素数p，下面同余式成立

$$C_n^m \equiv \prod_{i=0}^{k} C_{n_i}^{m_i} \pmod p$$

其中，$n_i, m_i (<= p-1)$分别为n, m的p进制表示的各个位

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$
$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$$

笼统点讲就是$C_n^m \equiv C_{n\%p}^{m\%p} C_{n/p}^{m/p} \pmod p$，这样我们只要对$C_{n/p}^{m/p}$再递归地调用$Lucas$定理就行了，所以代码其实挺好写

用法

适用条件

卢卡斯定理适合于数据范围较大但又不太大的情况，注意模数p需要是质数它可以把较大的组合数转化为多个较小的组合数乘积，但是如果模数p很大，n_i, m_i的数据范围($[0, p-1]$)也会很大，效果就会不好

怎么用

根据排列组合公式

$$C_n^m = \frac{n!}{m!(n-m)!}$$

我们可以先$O(n)$预处理出模p时阶乘数组$fact[n]$，为了$C_{n_i}^{m_i}\%p$，可以用快速幂求出$m!$和$(n-m)!$在模p意义下的乘法逆元，这样可以以$O(logn)$求出$C_{n_i}^{m_i}$

```
ll fac[maxn];
ll n, m, p;

// 快速幂就不再写一遍了
inline ll C(ll n, ll m, ll p) {
    if(n < m) return 0;
    return fac[n] * qpow(fac[m],p -2) * qpow(fac[n
- m], p - 2);
}

inline ll Lucas(ll n, ll m, ll p) {
    if(!m) return 1;
    return C(n % p, m % p, p) * Lucas(n / p, m /
p, p) % p;
```

```
}

int main() {
    scanf("%lld%lld", &n, &m, &p);
    fac[0] = fac[1] = 1L;
    for(ll i = 2L; i < p; i++)
        fac[i] = fac[i - 1] * i % p;
    printf("%lld\n", Lucas(n + m, m, p));
    return 0;
}
```

参考网站

[1] 学习笔记--数论--欧式线性筛素数

[2] 数论分块

[3] 扩展欧几里得算法详解

顺便推荐一下以上的几个博客，讲得通俗易懂