Math Theory II

Lyu Jiuyang

2021.4.24

Part 1 - 数论 II

同余（复习）

欧拉降幂（扩展欧拉定理）

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)}, & \gcd(a,p)=1 \\ a^b, & \gcd(a,p)\neq 1, b<\varphi(p) \\ a^{b \bmod \varphi(p)+\varphi(p)}, & \gcd(a,p)\neq 1, b\geq \varphi(p) \end{cases} \pmod p$$

例题

2019ICPC网络赛（南京）B.super_log 简化题意

计算 $2^{2^{2^{2^{\cdots}}}} \pmod p$

拓展中国剩余定理

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

（模数不互质）

例题

2019牛客暑期多校训练营（第十场） Han Xin and His Troop

高精度拓展中国剩余定理

思路：**求解 k 次扩展欧几里得**

设两个方程分别是 $x \equiv a_1 \pmod{m_1}$、$x \equiv a_2 \pmod{m_2}$

将它们转化为不定方程：$x = m_1 p + a_1 = m_2 q + a_2$，其中 p, q 是整数，则有 $m_1 p - m_2 q = a_2 - a_1$。

由裴蜀定理，当 $a_2 - a_1$ 不能被 $\gcd(m_1, m_2)$ 整除时，无解；其他情况下，可以通过扩展欧几里得算法解出来一组可行解 (p, q)；

则原来的两方程组成的模方程组的解为 $x \equiv b \pmod{M}$，其中 $b = m_1 p + a_1$，$M = \operatorname{lcm}(m_1, m_2)$，依次递推。

高次同余

BSGS算法

用于求 $a^x \equiv b \pmod p$ 高次方程的最小正整数解 x，其中 p 为素数。
设 $x = im - k$, 其中 $0 \leq k \leq m$。 那么方程变为

$$a^{im-k} \equiv b \pmod p$$

两边同乘 a^k

$$a^{im} \equiv a^k b \pmod p$$

先计算右边 $a^k b \ mod \ p$ 的值，把它放入一个hash表或是map里。再计算左边 $a^{im} \ mod \ p$ 的值，从小到大枚举所有可能的i值，再在hash表(map)查询。
复杂度 $O(\sqrt{p}))$

```cpp
// 超短的代码~
#include<bits/stdc++.h>
#define ll long long
using namespace std;
ll p,b,n,m;
map<ll,int>vis;
ll quick_pow(ll a,ll b,ll p){
}
int main(){
    scanf("%lld%lld%lld",&p,&b,&n);
    m=ceil(sqrt(p));
    for(ll i=0,t=n;i<=m;i++,t=t*b%p)vis[t]=i;
    for(ll i=1,tt=quick_pow(b,m,p),t=tt;i<=m;i++,t=t*tt%p)
    if(vis.count(t))printf("%lld\n",i*m-vis[t]),exit(0);
    puts("no solution");
    return 0;
}
```

矩阵乘法

矩阵快速幂

原理类似快速幂，中间使用矩阵乘法。
时间复杂度$O(m^3 \cdot logn)$

矩阵快速幂可用于计算斐波那契数列第 n 项。

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MOD=10000;
struct mat {
    ll a[2][2];
};
mat mat_mul(mat x,mat y) {
    mat res;
    memset(res.a,0,sizeof(res.a));
    for(int i=0;i<2;i++)
        for(int j=0;j<2;j++)
        for(int k=0;k<2;k++)
        res.a[i][j]=(res.a[i][j]+x.a[i][k]*y.a[k][j])%MOD;
    return res;
}
void mat_pow(int n) {
    mat c,res;
    c.a[0][0]=c.a[0][1]=c.a[1][0]=1;
    c.a[1][1]=0;
    memset(res.a,0,sizeof(res.a));
    for(int i=0;i<2;i++) res.a[i][i]=1;
    while(n)
    {
        if(n&1) res=mat_mul(res,c);
        c=mat_mul(c,c);
        n=n>>1;
    }
    printf("%lld\n",res.a[0][1]);
}
int main() {
    int n;
    while(~scanf("%d",&n)&&n!=-1) {
        mat_pow(n);
    }
    return 0;
}
```

高斯消元

原理

1. 增广矩阵行初等行变换为行最简形；
2. 还原线性方程组；
3. 求解第一个变量；
4. 补充自由未知量；
5. 表示方程组通解。

实现

https://blog.csdn.net/lzyws739307453/article/details/89816311

Part 2 - 组合数学

组合数的定义

从 n 个不同元素中，任取 m（$m \leq n$）个元素组成一个集合，叫做从 n 个不同元素中取出 m 个元素的一个组合；从 n 个不同元素中取出 m（$m \leq n$）个元素的所有组合的个数，叫做从 n 个不同元素中取出 m 个元素的组合数。用符号 C_n^m 来表示。

数学上用 $\binom{n}{m}$ 表示 C_n^m

$$\mathrm{C}_n^m = \frac{\mathrm{A}_n^m}{m!} = \frac{n!}{m!(n-m)!}$$

性质

在此介绍一些组合数的性质。

$$\binom{n}{m} = \binom{n}{n-m} \quad \text{（对称性）}$$

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} \quad \text{（定义）}$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \quad \text{（递推式）}$$

$$\binom{n}{0}+\binom{n}{1}+\cdots+\binom{n}{n}=\sum_{i=0}^{n}\binom{n}{i}=2^n$$ （横向求和）

$$\sum_{i=0}^{n}(-1)^i\binom{n}{i}=0$$ （带权横向求和）

$$\sum_{l=0}^{n}\binom{l}{k}=\binom{n+1}{k+1}$$ （斜向求和）

$$\sum_{i=0}^{m}\binom{n}{i}\binom{m}{m-i}=\binom{m+n}{m}\quad(n\geq m)$$

$$\binom{n}{r}\binom{r}{k} = \binom{n}{k}\binom{n-k}{r-k}$$

$$\sum_{k=1}^{m}\binom{m}{k}\binom{n}{k} = \binom{m+n}{m}$$

$$\sum_{i=0}^{n}\binom{n-i}{i} = F_{n+1}$$，F_n是斐波那契数列

求组合数

引理：卢卡斯定理

$$\binom{sp+q}{tp+r} = \binom{s}{t}\binom{q}{r}(mod\,p)，\ p为素数$$

则有

$$\binom{n}{m}\,mod\,p = \binom{n/p}{m/p}\binom{n\,mod\,p}{m\,mod\,p}\,mod\,p$$

复杂度$O(log_p n \cdot p)$

实现

1. 根据定义实现

复杂度$O(n)$。由于分子过大，int范围内只能计算到C(29,15)，如果只用阶乘只能计算到C(20,10)。

2. 杨辉三角打表

复杂度$O(n^2)$

```
for(i=0;i<n;i++)
    a[i][0]=a[i][i]=1;
for(i=2;i<n;i++)
    for(j=1;j<i;j++)
        a[i][j]=a[i-1][j-1]+a[i-1][j];
```

特别注意：C(30,15)=155117520，C(64,32)=1.83x10^18

3. 卢卡斯定理

复杂度$O(log_p n \cdot p)$，p是小素数（1e5）时使用。

```
ll qpow(ll a,ll n);
ll C(ll n,ll m){
    if(n<m) return 0;
    if(m>n-m) m=n-m;
    ll a=1,b=1;
    for(int i=0;i<m;i++){
        a=(a*(n-i))%p;
        b=(b*(i+1))%p;
    }
    return a*qpow(b,p-2)%p; //费马小定理求逆元
}
ll Lucas(ll n,ll m){
    if(m==0) return 1;
    return Lucas(n/p,m/p)*C(n%p,m%p)%p;
}
```

4. 预处理阶乘逆元表

使用定义式 `C(n,m)=n!/(m!*(n-m)!)`。

(1) 用O(n)的时间预处理逆元表 inv[n]。

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

(2) 预处理阶乘表 `fac[n]=(fac[n-1]*n)%p=(n!)%p`。

(3) 预处理阶乘的逆元表 `invfac[n]=(invfac[n-1]*inv[n])%p`。

```
void setFac(int n)
{
    fac[0]=facInv[0]=1;
    for (int i=1;i<=n;i++) {
        fac[i]=1LL*fac[i-1]*i%mod;
        facInv[i]=1LL*facInv[i-1]*inv[i]%mod;
    }
}
int C(int n,int m)
{
    if (n<m) return 0;
    if (n<0||m<0) return 0;
    int ans=fac[n];
    ans=1LL*ans*facInv[m]%mod;
    ans=1LL*ans*facInv[n-m]%mod;
    return ans;
}
```

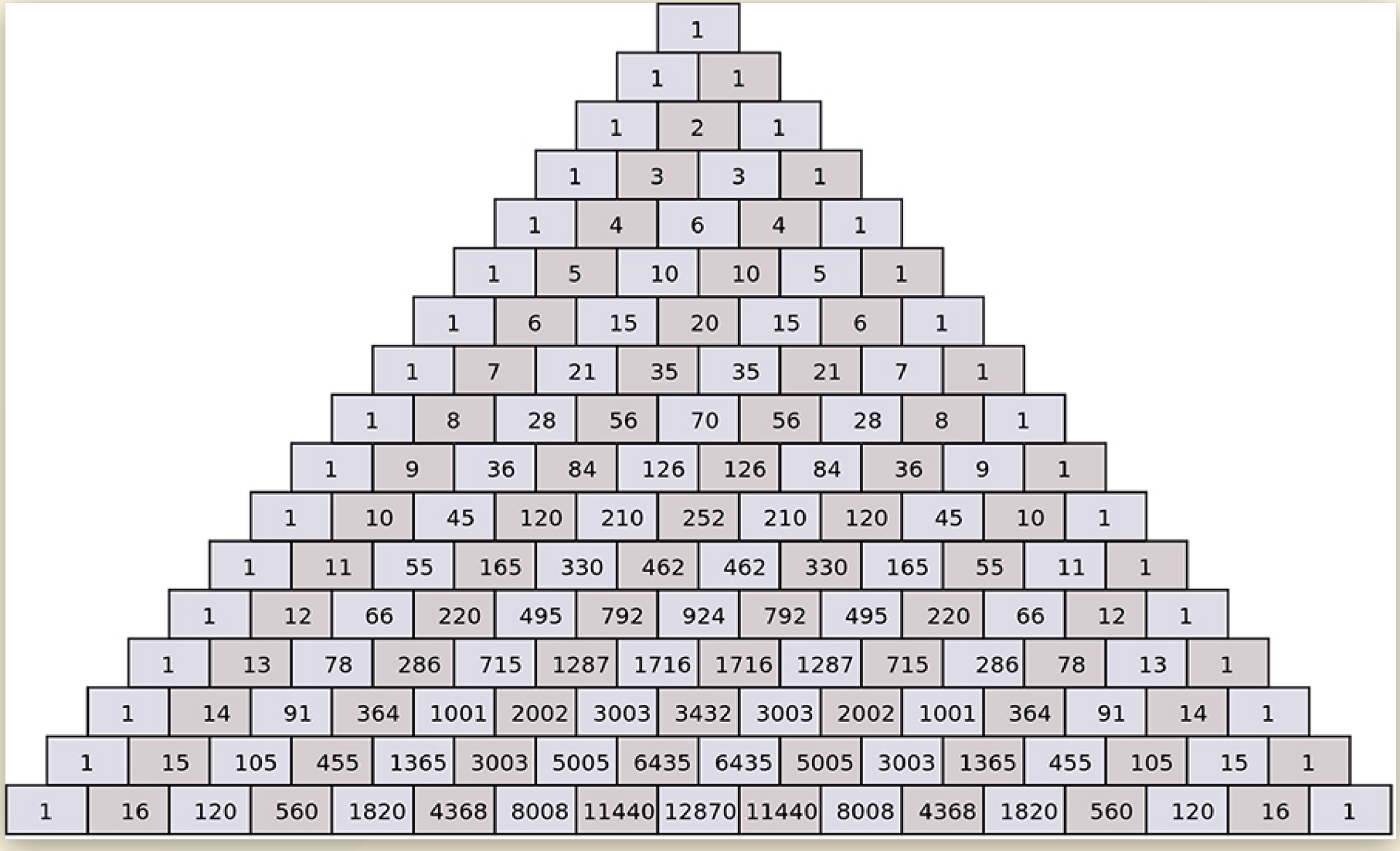
当小球遇上盒子

球是否相同？盒子之间是否相同？是否可以有空盒？

https://www.luogu.org/blog/chengni5673/dang-xiao-qiu-yu-shang-he-zi

更深入的学习可以自行搜索“母函数/生成函数”。

杨辉三角


1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1

错位排列

伯努利-欧拉装错信封问题

n 封不同的信，编号分别是 $1, 2, 3, 4, 5$ ，现在要把这 5 封信放在编号 $1, 2, 3, 4, 5$ 的信封中，要求信封的编号与信的编号不一样。问有多少种不同的放置方法?

“ 假设我们考虑到第 n 个信封，初始时我们暂时把第 n 封信放在第 n 个信封中，然后考虑两种情况的递推：
前面 $n-1$ 个信封全部装错；
前面 $n-1$ 个信封有一个没有装错其余全部装错。
对于第一种情况，前面 $n-1$ 个信封全部装错：因为前面 $n-1$ 个已经全部装错了，所以第 n 封只需要与前面任一一个位置交换即可，总共有 $f(n-1) \times (n-1)$ 种情况。
对于第二种情况，前面 $n-1$ 个信封有一个没有装错其余全部装错：考虑这种情况的目的在于，若 $n-1$ 个信封中如果有一个没装错，那么我们把那个没装错的与 n 交换，即可得到一个全错位排列情况。
其他情况，我们不可能通过一次操作来把它变成一个长度为 n 的错排。 ”

递推式

$$f(n) = (n-1)(f(n-1) + f(n-2))$$

其中，$f(1) = 0, f(2) = 1$。

公式

$$f_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + ... + (-1)^n \frac{1}{n!})$$

思考

部分错排（恰好有 k 个拿错）怎么办?

Part 3 - 数列

（承接秋季训练-简单数学与枚举约简）

斐波那契数列

斐波那契数列（The Fibonacci sequence）的定义如下：

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

该数列的前几项如下：

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

性质

斐波那契数列拥有许多有趣的性质，这里列举出一部分简单的性质：

- 卡西尼性质： $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$ 。
- 附加性质： $F_{n+k} = F_k F_{n+1} + F_{k-1}F_n$ 。
- 取上一条性质中 $k = n$ ，我们得到 $F_{2n} = F_n\left(F_{n+1} + F_{n-1}\right)$ 。
- 由上一条性质可以归纳证明， $\forall k \in \mathbb{N}, F_n \mid F_{nk}$ 。
- 上述性质可逆，即 $\forall F_a \mid F_b, a \mid b$ 。
- GCD 性质： $\left(F_m, F_n\right) = F_{(m,n)}$ 。
- 以斐波那契数列相邻两项作为输入会使欧几里德算法达到最坏复杂度。
- 齐肯多夫定理：任何自然数 n 可以被唯一地表示成一些互不相邻斐波那契数的和。

斐波那契数列通项公式

第 n 个斐波那契数可以在 $\Theta(n)$ 的时间内使用递推公式计算。但我们仍有更快速的方法计算。

解析解

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

这公式在计算的时侯要求极高的精确度，因此在实践中很少用到。但是请不要忽视！结合模意义下二次剩余和逆元的概念，在一些情况中中使用这个公式仍是有用的。

矩阵形式

斐波那契数列的递推可以用矩阵乘法的形式表达：

$$\begin{bmatrix} F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n-2} & F_{n-1} \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

设 $P = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$，我们得到

$$\begin{bmatrix} F_n & F_{n+1} \end{bmatrix} = \begin{bmatrix} F_0 & F_1 \end{bmatrix} \cdot P^n$$

于是我们可以用矩阵快速幂在 $\Theta(\log n)$ 的时间内计算斐波那契数列。

拓展

如果某个数列满足 $a_n = \alpha \cdot a_{n-1} + \beta \cdot a_{n-2}$，怎么求第100000000项呢?

例题

2020牛客寒假训练营 u's的影响力

“ $f(i) = f(i-1) \cdot f(i-2) \cdot a^b, f(1) = x, f(2) = y$，输出 $f(n)$，对1e9+7取模。$a, b, x, y, n < 10^{12}$ ”

找规律显然有 $f(n) = x^{F_{n-2}} y^{F_{n-1}} a^{F_{n-1} b}$

矩阵快速幂+快速幂+费马小定理求逆元解决.

2020 CCPC区域赛 - 威海 D. ABC Conjecture

定义函数 `rad(x)` 表示 x 所有质因子的单次幂的乘积

给定一个 c（1e18），询问能否找到 a 与 b 满足

$$a+b=c且c>rad(abc)$$

此处直接快进到结论：判断 c 能不能被素数的平方整除?

NOIP2011 山东省选 计算器

给定 y, z, p，计算 $y^z \bmod p$的值；
给定 y, z, p，计算满足 $xy \equiv z \pmod{p}$ 的最小非负整数 x；
给定 y, z, p，计算满足 $y^x \equiv z \pmod{p}$的最小非负整数 x。

($1 \leq y, z, p \leq 10^9$，p是质数)

清华大学2019校赛

在圆周上的n个点两两连线，最多可以把圆分成几份？

“ 找规律? 1,2,4,8,16,31,... ”

考虑在x个点（x>2）的图中新加入一个点，这个点会导致答案增加的数量为：$x-1+C_{x-1}^{3}$。

最终答案为 $1+\binom{n}{2}+\binom{n}{4}$（这里累加用组合数的哪个性质呢？）

参考

- OI-Wiki
- xyw5vplus1 师哥的《组合数学》课件
- 《数字魔鬼》(德) 汉斯·恩岑斯伯格 著
- 洛谷日报 & 洛谷题解
- Wikipedia

手动分割线。

以下部分可听可不听

Fast Fourier Transform (FFT)

快速傅里叶变换

概述

中文名：快速(离散)傅里叶变换

作用：以 $O(nlogn)$ 的复杂度计算多项式乘法

前置知识

多项式的系数表达和点值表达

系数表达：$F(x)=\sum_{i=0}^{n}a_ix^i$，如 $F(x)=x^2+x+1$

点值表达：$X=x_0,x_1,...x_n$ 代入多项式 $F(x)$，得到的 $n+1$ 个点分别为 $(x_0,y_0)(x_1,y_1)...(x_n,y_n)$

定理

“ 在平面直角坐标系中，$(n+1)$个点值对就能确定一个n次多项式的全部系数。 ”

比如，只要知道两个点的坐标就可以通过待定系数法确定之前的方程。

当次数比较高的时候，可以通过拉格朗日插值法进行确定。

也就是说我们计算两个 n 次多项式相乘只要取 $2n+1$个点进行分别计算就行了。

但是，如果随便取这些点，复杂度仍为 $O(n^2)$。

单位根

> 在复平面上，以原点为圆心，1为半径作圆，所得的圆叫单位圆。以圆点为起点，圆的n等分点为终点，做n个向量，设幅角为正且最小的向量对应的复数为ω_n，称为n次单位根。

在代数中，若$z^n = 1$，我们把z称为n次单位根。

性质

设n为2的整数次幂

1. $\omega_{2n}^{2k} = \omega_n^k$ （对应的向量相同）
2. $\omega_n^{k+\frac{n}{2}} = -\omega_n^k$ （对应的向量等大反向）

快速傅里叶变换

时间复杂度 $O(nlog_2n)$

理论

设函数

$$FL(x) = f_0 + f_2x + \ldots + f_{n-2}x^{n/2-1}$$

$$FR(x) = f_1 + f_3x + \ldots + f_{n-1}x^{n/2-1}$$

则有

$$F(x) = FL(x^2) + xFR(x^2)$$

代入ω_n^k、$\omega_n^{k+n/2}$，经计算化简

$$F(\omega_n^k) = FL(\omega_{n/2}^k) + \omega_n^k FR(\omega_{n/2}^k)$$

$$F(\omega_n^{k+n/2}) = FL(\omega_{n/2}^k) - \omega_n^k FR(\omega_{n/2}^k)$$

如果我们知道两个多项式 $FL(x)$ 和 $FR(x)$分别在$\omega_{n/2}^0, \omega_{n/2}^1, \omega_{n/2}^2, \ldots, \omega_{n/2}^{n/2-1}$的点值表示，就可以$O(n)$求出 $F(x)$在$\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$处的点值表示。
用分治法求下去。

把DFT中的 ω_n^1 换成 ω_n^{-1}，做完之后除以 n 即可实现IDFT。DFT/IDFT核心代码可以复用。

```
void fft(CP *f, bool flag) { // flag=-1为IDFT
  for (int p = 2; p <= n; p <<= 1) {
    int len = p >> 1;
    CP tG(cos(2 * Pi / p), sin(2 * Pi / p));
    if (!flag) tG.y *= -1;
    for (int k = 0; k < n; k += p) {
      CP buf(1, 0);
      for (int l = k; l < k + len; l++) {
        CP tt = buf * f[len + l];
        f[len + l] = f[l] - tt; // (1)
        f[l] = f[l] + tt; // (2)
        buf = buf * tG; //得到下一个[反]单位根
      }
    }
  }
}
```

关于IFFT的证明有很多，相对来说也比较复杂，这里给出一种数学上的理解。

$$\begin{pmatrix} x_0^0 & x_0^1 & x_0^2 & \cdots & x_0^{n-1} \\ x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^{n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n-1}^0 & x_{n-1}^1 & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix}$$

我们把上述矩阵抽象成 $WA = B$。

则 $A = W^{-1}B$

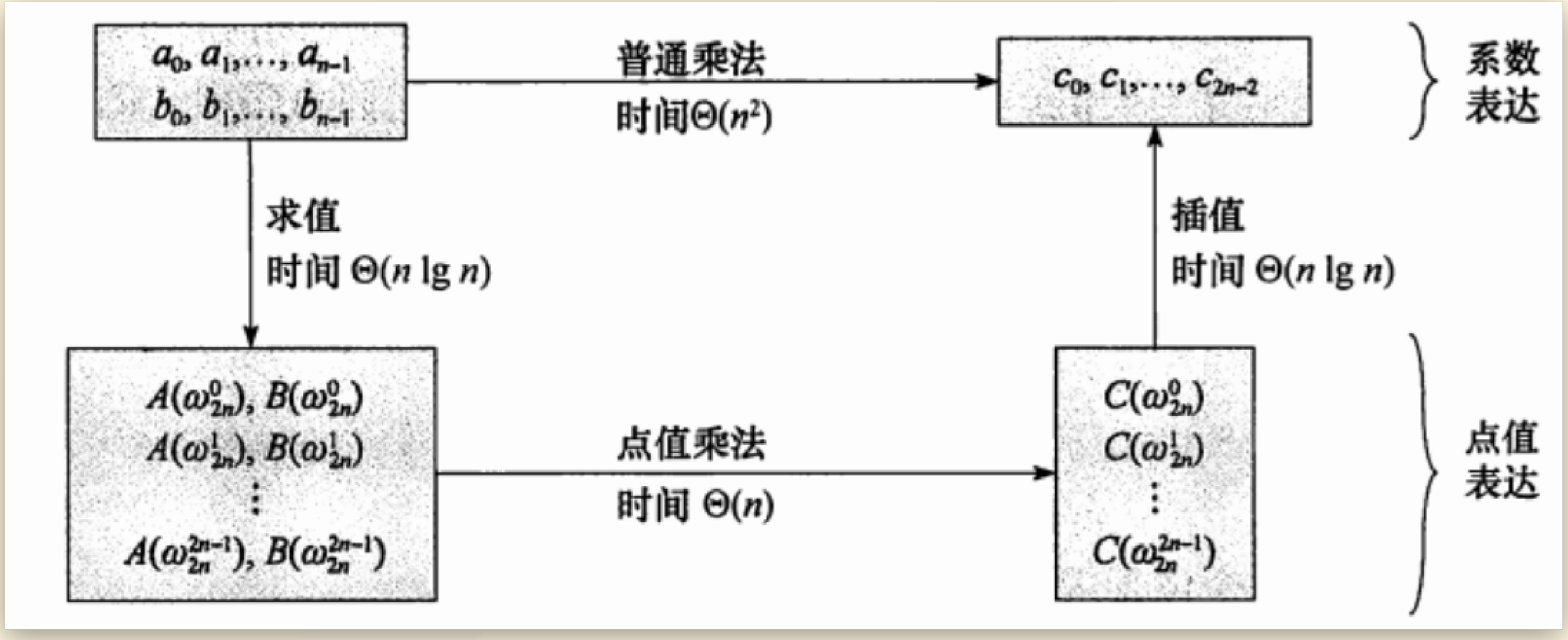
那么其实我们要做的就是范德蒙德矩阵求逆。

$$W=\begin{pmatrix}(w_n^0)^0 & (w_n^0)^1 & (w_n^0)^2 & \dots & (w_n^0)^{n-1}\\ (w_n^1)^0 & (w_n^1)^1 & (w_n^1)^2 & \dots & (w_n^1)^{n-1}\\ \dots & \dots & \dots & \dots & \dots\\ \dots & \dots & \dots & \dots & \dots\\ (w_n^{n-1})^0 & (w_n^{n-1})^1 & (w_n^{n-1})^2 & \dots & (w_n^{n-1})^{n-1}\end{pmatrix}=\begin{pmatrix}1 & 1 & 1 & \dots & 1\\ 1 & w_n^{1\times 1} & w_n^{1\times 2} & \dots & w_n^{1\times(n-1)}\\ \dots & \dots & \dots & \dots & \dots\\ \dots & \dots & \dots & \dots & \dots\\ 1 & w_n^{(n-1)\times 1} & w_n^{(n-1)\times 2} & \dots & w_n^{(n-1)\times(n-1)}\end{pmatrix}$$

$$W^{-1}=\frac{1}{n}\begin{pmatrix}1 & 1 & 1 & \dots & 1\\ 1 & w_n^{-1\times 1} & w_n^{-1\times 2} & \dots & w_n^{-1\times(n-1)}\\ \dots & \dots & \dots & \dots & \dots\\ \dots & \dots & \dots & \dots & \dots\\ 1 & w_n^{-(n-1)\times 1} & w_n^{-(n-1)\times 2} & \dots & w_n^{-(n-1)\times(n-1)}\end{pmatrix}$$

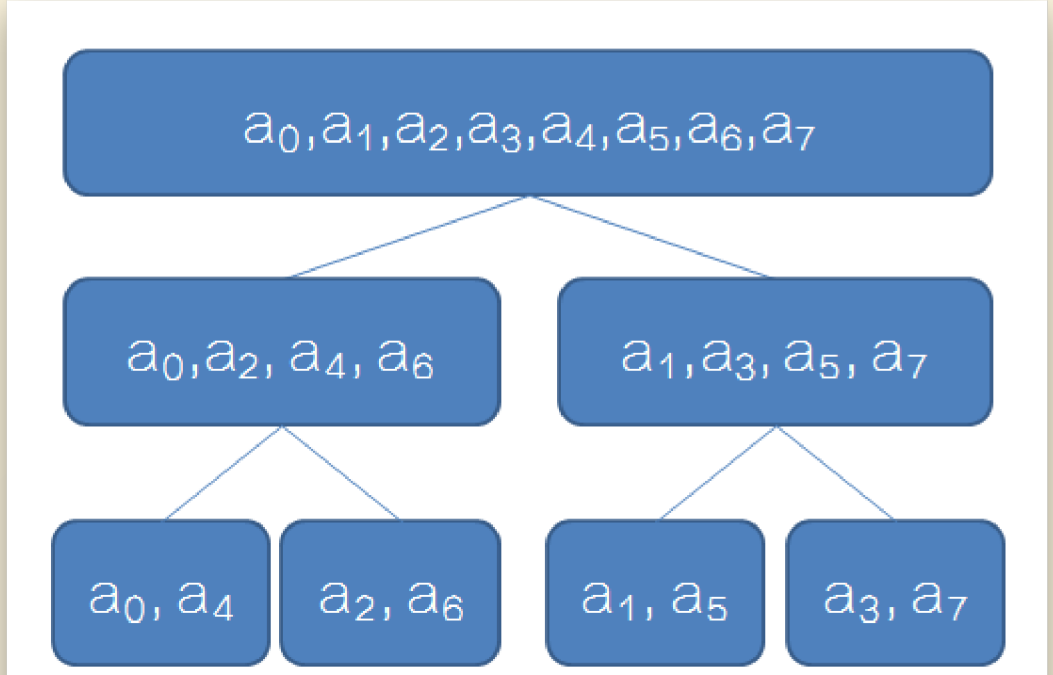
观察发现，只需要变换指数的正负，再乘 $\frac{1}{n}$，IFFT和FFT的流程就完全一致了。

点值表示和系数表示之间的转换



优化：蝴蝶变换

要求的序列实际是原序列下标的二进制反转。



原序列:	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
后序列:	0	4	2	6	1	5	3	7
	000	100	010	110	001	101	011	111

优化：三次变两次

根据 $(a+bi)*(c+di) == ac-bd+adi+bci$

要求 $F(x)*G(x)$，设复多项式 $P(x)=F(x)+G(x)i$,

则 $P(x)^2 == F(x)^2-G(x)^2+2F(x)G(x)i$

发现$P(x)^2$的虚部为 $2F(x)G(x)$

也就是说求出 $P(x)^2$之后,把它的虚部除以2即可。

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
#define Maxn 1350000
using namespace std;
const double Pi = acos(-1);
inline int read() {
  register char ch = 0;
  while (ch < 48 || ch > 57) ch = getchar();
  return ch - '0';
}
int n, m;
struct CP {
  CP(double xx = 0, double yy = 0) { x = xx, y = yy; }
  double x, y;
  CP operator+(CP const &B) const { return CP(x + B.x, y + B.y); }
  CP operator-(CP const &B) const { return CP(x - B.x, y - B.y); }
  CP operator*(CP const &B) const {
    return CP(x * B.x - y * B.y, x * B.y + y * B.x);
  }
} f[Maxn << 1];  //只用了一个复数数组
int tr[Maxn << 1];
void fft(CP *f, bool flag) {
  for (int i = 0; i < n; i++)
    if (i < tr[i]) swap(f[i], f[tr[i]]);
  for (int p = 2; p <= n; p <<= 1) {
    int len = p >> 1;
    CP tG(cos(2 * Pi / p), sin(2 * Pi / p));
    if (!flag) tG.y *= -1;
    for (int k = 0; k < n; k += p) {
      CP buf(1, 0);
      for (int l = k; l < k + len; l++) {
        CP tt = buf * f[len + l];
        f[len + l] = f[l] - tt;
        f[l] = f[l] + tt;
        buf = buf * tG;
      }
    }
  }
}
int main() {
  scanf("%d%d", &n, &m);
  for (int i = 0; i <= n; i++) f[i].x = read();
  for (int i = 0; i <= m; i++) f[i].y = read();
  for (m += n, n = 1; n <= m; n <<= 1);
  for (int i = 0; i < n; i++)
    tr[i] = (tr[i >> 1] >> 1) | ((i & 1) ? n >> 1 : 0);
  fft(f, 1);
  for (int i = 0; i < n; ++i) f[i] = f[i] * f[i];
  fft(f, 0);
  for (int i = 0; i <= m; ++i) printf("%d ", (int)(f[i].y / n / 2 + 0.49));
  return 0;
}
```

其他

尽管FFT优于朴素算法，但是由于常数过大（复数背后是浮点数运算），同时还有精度的限制，所以数论中仍有其他的算法如NTT（快速数论变换），供大家后续学习。

例题

洛谷P3803 多项式乘法

参考资料

FFT学习笔记
快速傅里叶变换详解