# **[多项式相关（求逆、开根、除法、取模）](https://www.cnblogs.com/yoyoball/p/8724115.html)**

## **多项式求逆（元）**

* 定义

　　对于一个多项式A(x)，如果存在一个多项式B(x)，满足B(x)的次数小于等于A(x)且A(x)B(x)≡1(mod xn)，那么我们称B(x)为A(x)在模xn意义下的逆元，简单记作A−1(x)

* 求解

　　从最简单的情况开始考虑，当n=1的时候A(x)≡ c (mod x)，c为A(x)的常数项，此时A−1(x)为c−1的逆元

　　在这个基础上我们继续考虑一般情况

　　对于n>1的情况，不妨设B(x)=A−1(x)，那么我们可以根据定义列出下面的式子：

A(x)B(x)≡1(mod xn)

　　这里的话考虑用倍增的方式求解（算倍增吧），这里假设我们已经知道了A(x)在mod x⌈n/2⌉下的逆元G(x)，那么有：

A(x)G(x)≡1(mod x⌈n/2⌉)

　　我们把A(x)和B(x)的式子写成mod x⌈n/2⌉下的：

　　（可以这么写是因为mod xn相当于将乘积中x次数大于等于n的忽略掉了，而mod x⌈n/2⌉则相当于忽略了更多的项，既然前者满足，那么后者肯定也满足）

A(x)B(x)≡1(mod x⌈n/2⌉)

　　把这两条式子相减，就可以搞事情了：

A(x)[B(x)−G(x)]≡0 (mod x⌈n/2⌉)

B(x)−G(x)≡0 (mod x⌈n/2⌉)

　　然后我们两边平方一下：

B2(x)−2B(x)G(x)+G2(x)≡0 (mod x⌈n/2⌉)

　　然后这里有个很神奇的事情，B(x)−G(x) 在mod x⌈n/2⌉下为0，说明这个式子最后的结果的0到⌈n/2⌉−1次项系数都为0，平方了之后，对于结果的i次项系数，（0<=i<=2∗⌈n/2⌉−1 ），其系数，而aj和ai−j中必定有一项为0（因为j和i−j中必定有一个值小于⌈n/2⌉），所以我们可以得到一个结论，这个式子在平方了之后在mod xn下也是0

　　那么我们就可以写成：

B2(x)−2B(x)G(x)+G2(x)≡0 (mod xn)

A(x)B(x)∗B(x)−2∗A(x)B(x)∗G(x)+A(x)G2(x)≡0 (mod xn)

　　两边同时乘上A(x)，由逆元的定义我们可以将上面的式子化简成下面这样：

B(x)−2G(x)+A(x)G2(x)≡0 (mod xn)

　　最后得到：

**B(x)≡2G(x)−A(x)G2(x) (mod xn)**

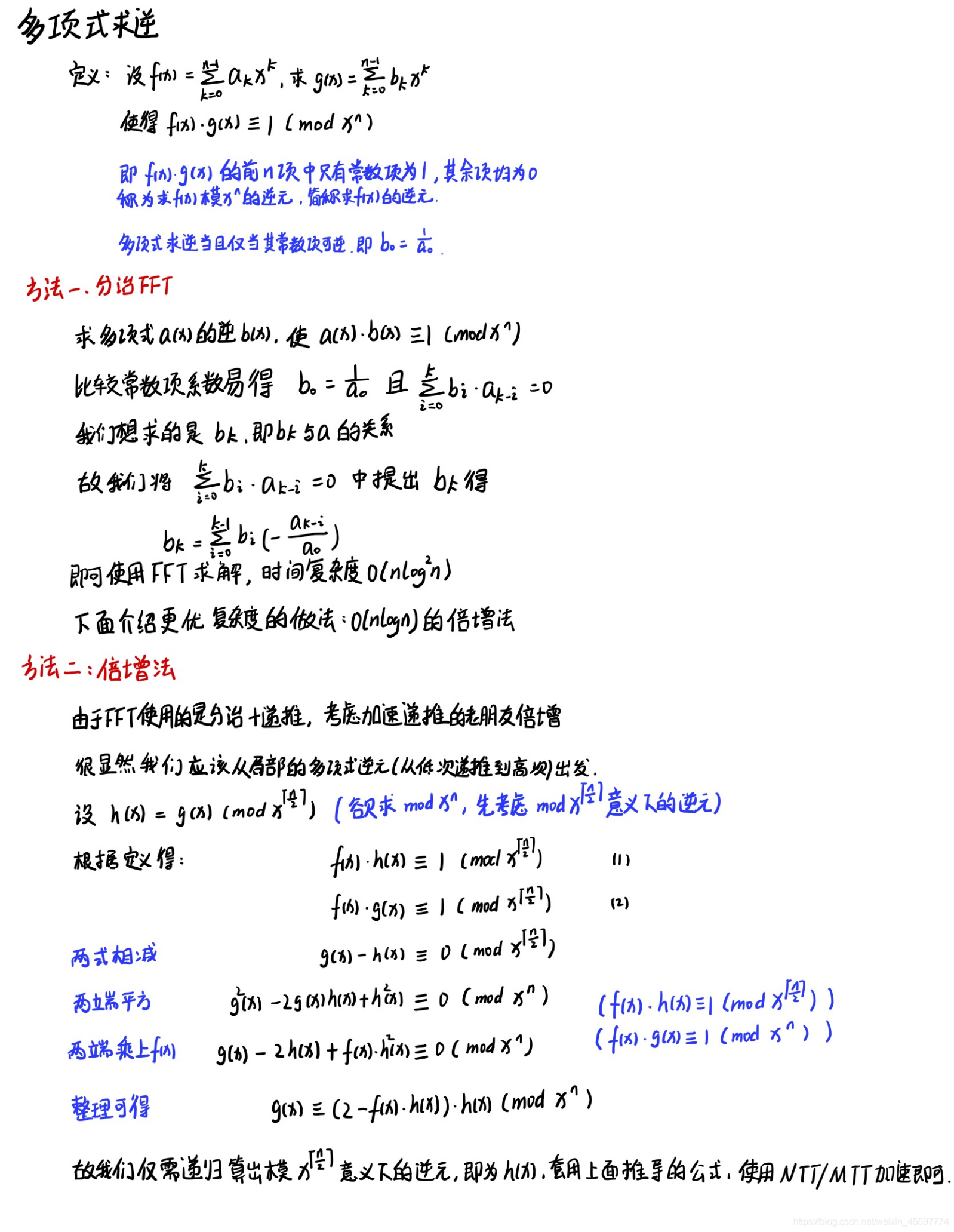
　　也就是说，如果我们知道G(x)，我们就可以推出B(x)了

　　具体的实现可以用递归的方式实现，中间的多项式乘法可以用fft加速一下，那么最终的时间复杂度就是

T(n)=T(n2)+O(n log n)=O(n log n)

　　然而为啥这样搞完了还是一个log呢？因为每次递归下去多项式的最高次数都会减半，稍微算一下就会发现最后总的时间复杂度合起来还是一个log而不是两个

注意，后面这一堆推式子的过程是建立在n=1的时候有解的前提下的，所以我们还可以得到一个结论：一个多项式在mod xn下是否有逆元取决于其常数项在mod xn下是否有逆元



* 实现

[模板：洛谷P4238](https://www.luogu.org/problemnew/show/P4238)

#include<algorithm>

#include<iostream>

#include<cstring>

#include<cstdio>

#include<cmath>

#include<map>

#define Redge(u) for (int k=h[u],to; k; k=ed[k].nxt)

#define REP(i,n) for (int i=1; i <= (n); i++)

#define mp(a,b) make\_pair<int,int>(a,b)

#define cls(s) memset(s,0,sizeof(s))

#define cp pair<int,int>

#define LL long long int

using namespace std;

const int maxn=400005,maxm=100005,INF=1000000000;

inline int read(){

int out=0,flag=1; char c=getchar();

while (c<48 || c > 57){

if (c == '-') flag=-1;

c=getchar();

}

while (c >= 48 && c <= 57){

out=(out<<3)+(out<<1) + c-48;

c=getchar();

}

return out\*flag;

}

const int G=3,P=998244353;

inline int qpow(int a,int b){

int re=1;

for (; b; b >>= 1,a=1ll \* a \* a % P)

if (b&1) re=1ll \* re \* a % P;

return re;

}

int a[maxn],b[maxn],c[maxn],R[maxn];

void NTT(int\* a,int n,int f){

for(int i=0; i<n; i++) if(i<R[i]) swap(a[i],a[R[i]]);

for(int i=1; i<n; i <<= 1){

int gn=qpow(G,(P-1)/(i<<1));

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

int g=1,x,y;

for (int k=0; k<i; k++,g=1ll\*g\*gn%P){

x=a[j+k]; y=1ll\*g\*a[j+k+i] % P;

a[j+k]=(x+y)%P; a[j+k+i]=((x-y)%P+P)%P;

}

}

}

if(f==1) return;

int nv=qpow(n,P-2);

reverse(a+1,a+n);//交换法，因为g^(-1)=g^(n-1)，所以可以先正变换，翻转就为逆变换。

for (int i=0; i<n; i++) a[i]=1ll\*a[i]\*nv%P;

}

void inv(int deg,int\* a,int\* b){

 if (deg == 1){b[0]=qpow(a[0],P-2); return;}

inv((deg + 1)>>1,a,b);

int L=0,n=1;

while (n<(deg<<1)) n <<= 1,L++;

for (int i=1; i<n; i++) R[i]=(R[i>>1]>>1) | ((i&1)<<(L-1));

for (int i=0; i<deg; i++) c[i]=a[i];

for (int i=deg; i<n; i++) c[i]=0;

NTT(c,n,1); NTT(b,n,1);

for (int i=0; i<n; i++)

b[i]=1ll\*((2ll-1ll\*c[i]\*b[i]%P)+P)%P\*b[i]% P;

NTT(b,n,-1);

for (int i=deg; i<n; i++) b[i]=0;

}

int main(){

int n=read();

for (int i=0; i<n; i++) a[i]=read();

inv(n,a,b);

for (int i=0; i<n; i++) printf("%d ",b[i]);

return 0;

}

## **多项式开根**

* 定义

　　对于一个多项式A(x)，如果存在一个多项式B(x)，满足B2(x)≡ A(x)(mod xn)，则称B(x)为A(x)在mod xn下的平方根

* 求解

　　同样是考虑最简单的情况，当n=0的时候，B(x)的常数项就是1

　　然后考虑一般情况，同样的思路，考虑用倍增的方式来求

　　假设我们已经知道了A(x)在mod xn下的平方根G(x)，现在要求在mod x2n下的平方根B(x)，根据定义我们可以列出式子：

B2(x)≡A(x)(mod x2n)

G2(x)≡A(x)(mod xn)

　　我们对这个式子进行一些处理：

G2(x)≡A(x)(mod xn)

G2(x)−A(x)≡0(mod xn)

　　那么可以得到（因为右边是0所以可以这么搞）：

(G2(x)−B2(x))2≡0(mod x2n)

G4(x)−2G2(x)B2(x)+B4(x)≡0(mod x2n)

　　然后两边加上4G2(x)B2(x)：

G4(x)+2G2(x)B2(x)+B4(x)≡4G2(x)B2(x) (mod x2n)

(G2(x)+B2(x))2≡4G2(x)B2(x)(mod x2n)

(G2(x)+B2(x))2≡(2G(x))2B2(x)(mod x2n)

(G2(x)+A(x))2≡(2G(x))2B2(x)(mod x2n)

我们将(2G(x))2移到左边去，将左边写成一个平方的形式，得到：

等式左边的东西就是我们要求的B(x)



所以如果说我们知道了G(x)，我们也就可以得出B(x)，分母可以用多项式求逆搞一下，其他的多项式乘法fft搞一下，问题不大

　　总的复杂度是：

T(n)=T(n/2)+求逆复杂度+O(nlog n)=O(n log n)

* 实现

　　namespace NTT中主要需要调用的过程长这个样子

void Ntt\_getsqrt(vct &a,vct &invb,int n,int m){

prework(a,invb,n,m);

ntt(A,1);

ntt(B,1);

for (int i=0;i<len;++i)

B[i]=1LL\*B[i]\*inv2%MOD\*A[i]%MOD;

ntt(B,-1);

}

　　开根的话大概长这个样子

vct Sqrt(vct a){

int N=a.size(),M,M1;

if (N==1){

a[0]=1;

return a;

 }

vct b=a,invb;

b.resize((N+1)>>1);

b=Sqrt(b);

invb=b; invb.resize(N);//resize!!!

invb=Inv(invb);

NTT::Ntt\_getsqrt(a,invb,N,N);

b.resize(NTT::len);

for (int i=0;i<NTT::len;++i) b[i]=(1LL\*b[i]\*inv2%MOD+NTT::B[i])%MOD;

b.resize(N);

return b;

}

## **多项式除法**

* 问题

　　给出一个n次多项式A(x)，以及一个（m（m<=n)次多项式B(x)

　　要求出D(x)满足A(x)=D(x)B(x)+R(x)，且D(x)的次数<=n−m，R(x)的次数<m

　　简单来说就是类比整数的除法，D(x)就是商，R(x)就是余数，我们现在考虑求商

* 求解

　　为了方便接下来的表述，先定义一些操作，我们记：

rev(A(x))=xnA(1/x)

　　也就是系数反转，举个简单的例子：

A(x)=4x4+3x3+2x2+1

rev(A(x))=x4+2x3+3x2+4

　　那么现在我们把上面那条式子搬下来：

A(x)=D(x)B(x)+R(x)

　　（接下来的步骤均将D(x)看成n−m次多项式，R(x)看成m−1次多项式，对于那些不存在的高次项我们就把系数看成0就好了）

　　后面的余数看起来十分不友善，所以我们要想个办法把它去掉，于是我们可以进行以下的操作:

　　我们将上面式子中的所有x换成1/x，然后等式两边同时乘上xn，得到：

xnA(1/x)=xn−mD(1/x)xmB(1/x)+xn−m+1xm−1R(1/x)

rev(A(x))=rev(D(x))rev(B(x))+xn−m+1rev(R(x))

　　现在再来看一下各个项的最高次项，首先是我们要求的元素D(x)，由于这个多项式原来是n−m次，所以在系数反转之后肯定不会超过n−m次，而我们要“消掉”的R(x)原来是m−1次多项式，所以xn−m+1R(x)的**最低次项**应该是**大于**n−m 的

　　那么考虑将上面的式子放到mod xn−m+1下，xn−m+1R(x)的影响就可以十分愉快滴被消掉啦，同时我们也不会影响到D(x)的求解，因为D(x)是n−m次的。

　　于是我们就得到了这样一个式子：

rev(A(x))≡ rev(D(x))rev(B(x)) (mod xn−m+1)

　　那所以，我们只要求一个rev(B(x))在mod xn−m+1意义下的逆元然后跟rev(A(x))乘一下，得到rev(D(x))，然后再把系数反转回来就得到D(x)啦

* 实现

　　除法大概是长这个样子

vct operator / (vct a,vct b){

int N=a.size()-1,M=b.size()-1;

if (N<M){

d.resize(1);d[0]=0;

return d;

}

reverse(a.begin(),a.end());

reverse(b.begin(),b.end());

b.resize(N-M+1);

d=Inv\_p(b)\*a;

d.resize(N-M+1);

reverse(d.begin(),d.end());

return d;

}​

## **多项式取模**

* 问题

　　 这个。。其实就是求上面那个R(x)

* 求解

　　有了多项式除法（也就是求商）之后，求余数就变得比较简单了

　　类比整数的取模，我们可以得到这样的一个式子：

R(x)=A(x)−D(x)B(x)

　　那就除法求出D(x)之后直接减一下就好了，D(x)B(x)这个多项式乘法也是直接用fft求就好了

* 实现

void mod(vct &a,vct b){

int N=a.size()-1,M=b.size()-1;

if (N<M) return;

t=a/b;

a=a-(t\*b);

a.resize(M);

}