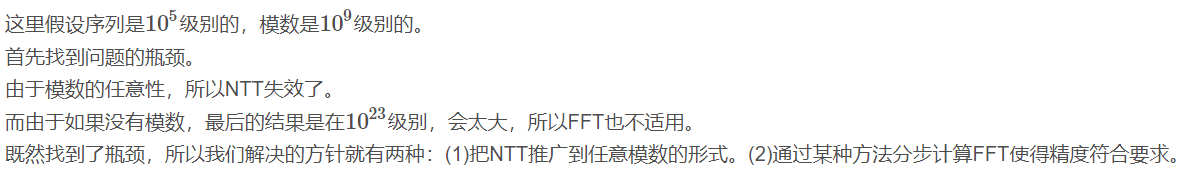
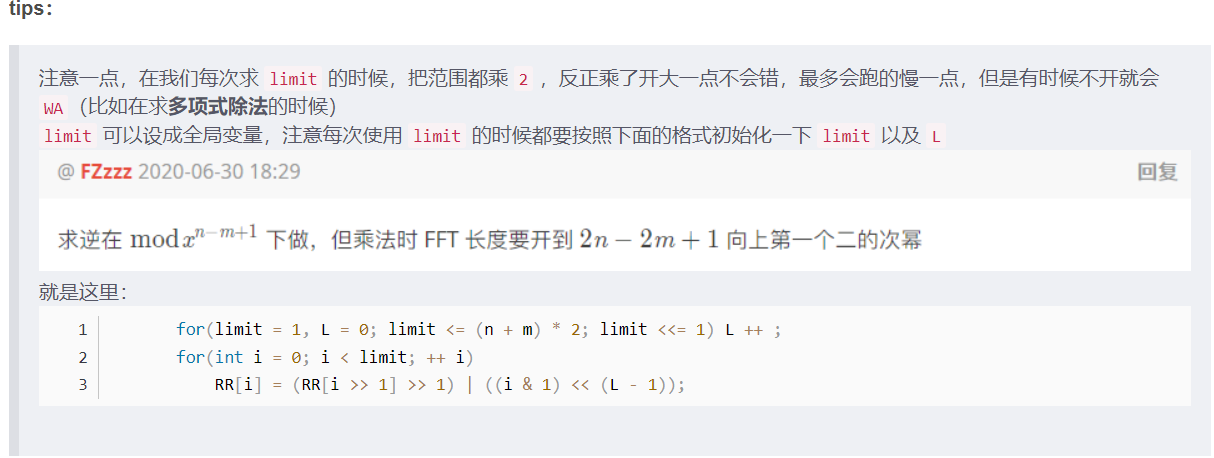


一般fft用的是double，精度为16位，而题目需要的精度一般为n\*max(fi,gi)^2,下面是一个例子，所以有的时候会超过double，这个时候可以改用long double



下面这个是记得每次fft之前都要初始化一些东西，即st()函数



#include<bits/stdc++.h>

#define ll long long

#define inf 1e18

#define mn 5000005//最好开到(n+m)的2.5倍左右，至少要开2倍

using namespace std;

const double pi=acos(-1);

ll n,m,res,ans[mn];

ll lim;//补齐的2的整数幂N

ll maxl;//二进制的位数

ll r[mn];//二进制翻转

struct com

{

double x,y;

com(double x=0,double y=0):x(x),y(y){}

}a[mn],b[mn];

com operator \* (com j,com q){return com(j.x\*q.x-j.y\*q.y,j.x\*q.y+j.y\*q.x);}//复数乘法：模长相乘，幅度相加

com operator - (com j,com q){return com(j.x-q.x,j.y-q.y);}

com operator + (com j,com q){return com(j.x+q.x,j.y+q.y);}

void fft(com \*a,ll type)//fft板子

{ll i,mid,k,len,pos;

for(i=0;i<lim;i++)

if(i<r[i])swap(a[i],a[r[i]]);

for(mid=1;mid<lim;mid<<=1)

{

com wn(cos(pi/mid),type\*sin(pi/mid));//使用初始化函数的办法

for(len=mid<<1,pos=0;pos<lim;pos+=len)

{

com w(1,0);//使用初始化函数的办法

for(k=0;k<mid;k++,w=w\*wn)

{

com x=a[pos+k];

com y=w\*a[pos+mid+k];

a[pos+k]=x+y;

a[pos+mid+k]=x-y;

}

}

}

if(type==1)return ;

for(i=0;i<=lim;i++){a[i].x/=lim;a[i].y/=lim;}

}

void cal(com \*a,ll deg)

{ll i;

for(lim=1,maxl=0;lim<=deg;lim<<=1)maxl++;

for(i=0;i<lim;i++)

r[i]=(r[i>>1]>>1)|((i&1)<<(maxl-1));

fft(a,1);

for(i=0;i<=lim;i++)

a[i]=a[i]\*a[i];//求出a(x)^2

fft(a,-1);

}

int main()

{

ll x,y,z,i,j,k;

while(~scanf("%lld%lld",&n,&m))

{

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

scanf("%lf",&a[i].x);

for(i=0;i<=m;i++)//这个是三步化两步的优化版本，正常版本见公用板子里面

scanf("%lf",&a[i].y);//把b(x)放到a(x)的虚部上

cal(a,n+m);

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

printf("%lld ",(ll)(a[i].y/2+0.5));

//虚部取出来除2，注意要+0.5，否则精度会有问题,这里的x和y都是double

printf("\n");

for(i=0;i<=lim;i++)

a[i].x=a[i].y=0;

}

}

很显然，fft也可以用来处理a\*b的高精度版本，只需要把每一位上的数当成是多项式系数，最后再加上进位就好了，也就是：

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

{

//取实数四舍五入，此时虚数部分应当为0或由于浮点误差接近0

ans[i]+=(int)(a[i].real()+0.5);

ans[i+1]+=ans[i]/10;

ans[i]%=10;

}

while(!ans[limit]&&limit>-1)limit --;

if(limit==-1)printf("0");

else

for(int i=limit;i>=0;i--)

printf("%d",ans[i]);