# 比FFT还容易明白的NTT（快速数论变换）

# **NTT相关**

一种****快速数论变换算法****，这种算法是以数论为基础，对样本点为N=2r的数论变换，按时间抽取的方法，得到一组等价的迭代方程，有效高速简化了方程中的计算公式·与直接计算相比，大大减少了运算次数。（见快速傅里叶变换）。  
在计算机实现多项式乘法中，我们所熟知的快速傅里叶变换(FFT)是基于n次单位根 (omega) 的优秀性质实现的，而由于其计算时会使用正弦函数和余弦函数，在不断运算时无法避免地会产生精度误差。而多项式乘法有些时候会建立在模域中，在对一些特殊的大质数取模时，便可以考虑用原根g来代替 ，而这些特殊的大质数的原根恰好满足 的某些性质，这使得多项式乘法在模域中也可以快速的分治合并。  
——百度百科

****NTT（Number Theoretic Transform）****，中文名****快速数论变换****  
和FFT一样，NNT也用来****加速多项式乘法****，不过NNT最大的优点是****可以取模****  
或者可以理解为NNT是FFT取模升级版

* ****看这篇****NNT****之前确保你已经会了****FFT
* 好像NNT比起FFT来难的知识点更少了

# **NTT的优缺点**

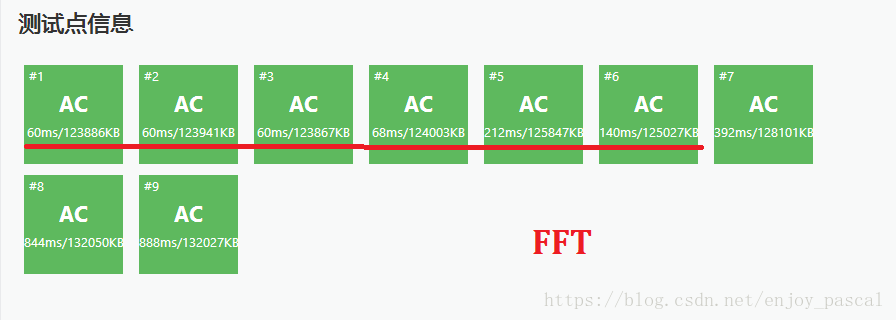
## **优点**

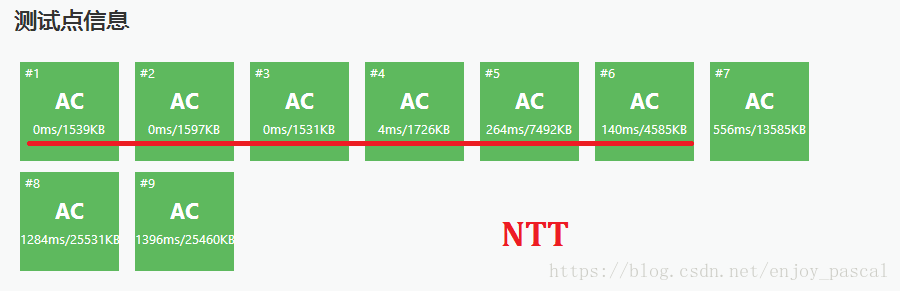
****能取模****，FFT的复数你给我来取个模

****没有精度差****，FFT浮点数的精度怎么也会出一点问题

由于均为整数操作（虽然取模多），NNT常数小，****通常****比一大堆浮点运算的FFT要****快****







我只能说NNT****小数据下表现良好…****

## **缺点**

多项式的系数都必须是****整数****

****模数有限制****，NNT题的模数通常都是****相同****的998244353

其实这些模数的****原根****通常都是3

# **NTT前置知识&技能**

## **原根**

对于g,p∈Z，如果gi mod p(1⩽i⩽p−1)的值互不相同，则称g为p的原根  
或者说∀i,j(1⩽i,j⩽p−1,i<j),gi mod p≠gj mod p，那么g为p的原根  
原根没什么快速求法，只能暴力枚举判断  
通常模数常见的有998244353,1004535809,469762049，这几个的原根都是3

# **NTT（快速数论变换）**

FFT可以大大优化是因为ω有着神奇且优秀的性质  
****其实原根也有这种性质！****

在NNT里，我们可以拿原根来****代替****FFT的单位根  
具体就是，当合并区间的长度为len=2\*mid时，单位根为而原根即为​  
注意大多题目的模数p=998244353，此时g=3，即可代入原根计算NNT了  
如果理解了FFT的话，NNT也就迎刃而解了  
****时间复杂度****O(nlog2n)

int temp=pow(g,(mod-1)/(mid\*2));//用原根代替单位根

# **NTT板子**

这些是定义

#define g 3//模数的原根

#define mod 998244353//通常情况下的模数

int pow(int x,int y)//快速幂

{

ll z=1ll\*x,ans=1ll;

for (;y;y/=2,z=z\*z%mod)if (y&1)ans=ans\*z%mod;//注意精度

return (int)ans%mod;

}

这个是NNT板子，拿FFT的改那么几下就好了

void ntt(int a[],int inv)

{

int bit=0;

while ((1<<bit)<n)bit++;

fo(i,0,n-1)

{

rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));

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

}//前面的东西和FFT一样

for (int mid=1;mid<n;mid\*=2){//倍增次数

 int temp=pow(g,(mod-1)/(mid\*2));//用原根代替单位根

if (inv==-1)temp=pow(temp,mod-2);//逆变换则乘上逆元

for (int i=0;i<n;i+=mid\*2){//i，被合并的第1个数

int omega=1;

for (int j=0;j<mid;j++,omega=1ll\*omega\*temp%mod){//j,计算y值个数

int x=a[i+j],y=1ll\*omega\*a[i+j+mid]%mod;

a[i+j]=x+y,a[i+j+mid]=x-y;

a[i+j]=(a[i+j]%mod+mod)%mod,a[i+j+mid]=(a[i+j+mid]%mod+mod)%mod;//注意取模

}

}//大体流程和FFT差不多

}

**if(inv==-1){**

**int Inv=pow(n,mod-2);**

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

**a[i]=(LL)a[i]\*Inv%mod;**

**}**

}

和FFT一样，NNT也只能处理n为2的次幂的多项式  
NNT调用和FFT一模一样，注意longlong和除法都变成乘逆元  
逆NTT变换后每一项系数乘上模数p的****逆元****

# 快速数论变换(NTT)——学习笔记：<https://blog.csdn.net/chhnz/article/details/79293783>

<https://zhuanlan.zhihu.com/p/80297169>