# $NTT$ 学习笔记

用了差不多一整天来学 $FFT$ ，有的算法是学会了之后就觉得很简单，甚至不知道为什么之前看不懂，但这是一个我学会了之后还是觉得很难的算法 _(:з」∠)_

因为 $NTT$ 是在 $FFT$ 的基础上学的，好像要简单一点点。

- $FFT$ 的缺点
- $NTT$ 的优点
- 两者的比较

## $FFT$ 的缺点

如果说 $FFT$ 有什么缺点，那就是明明是整数多项式却非要用 $double$ 存，算着算着，还算出虚数来了...精度问题还是要注意一下的。

## $NTT$ 的优点

$NTT$ 非常的神奇！它可以省去复数的运算，完全在整数域中进行运算。要知道为什么，让我们回到最初关于单位根的部分找找答案。

> 看看这个方程 $x^n=1$
>
> 这个方程有几个解？看这篇文章之前我们知道，如果n是0，那么有无数个，如果n是偶数，有$1$，$-1$，否则就只有 $1$ 了。
>
> 但是现在有了复数这种奇妙的东西，答案就不同了。

这是从我自己的文章里引用的话...

拓宽一下思路，还有什么数是满足这个方程的？

> 提示：从 $OI$ 的角度想。

如果就是想找到一个大于一的数，但是它的 $n$ 次方等于 $1$ ....取模啊。

想到这里，事情就变得简单了，只要找到一些这样的 "单位根“ ，就可以代替复数单位根进行运算了，听起来非常的 $exciting$ .

定义一个新的符号 $\delta_p(a)$ ，虽然我的印象中 $delta$ 是那个小三角来着.....，但是这个符号的latex公式才叫 $delta$ 。~~我复读我自己~~

它的意思是，对于 $a,p$，$(a,p)=1$ ，找到最小的 $k$ ，使得 $a^k\equiv 1\ (\%p)$，那么 $k$ 就称为 $a$ 模 $p$ 的阶，记作 $\delta_p(a)$ 。由于欧拉定理，可以得到 $\delta_p(a)|\varphi(p)$ ，对于某些特殊的数，$\delta_p(a)=\varphi(p)$ ，此时称 $a$ 为模 $p$ 的一个原根。

找原根的目的还是为了代入多项式，所以依旧需要找多个单位根。首先找一个合适的质数，满足$p=a2^n+1$ ，这里的 $a,n$ 都是普通的整数，定义原根为 $g$ ，$\omega_n=g^a$ 。一般来说，选择的质数会是 $998244357$ ，它的原根是 $3$。

之前的单位根有一些独特的性质，现在的单位根还有吗？(以下内容多为搬运)

> 1. $\omega_{2n}^{2k}=\omega_n^k$

把 $p$ 拆掉，变成 $p = \frac{a}{2} \cdot 2n + 1$，就有 $\omega_{2n} = g^{\frac{a}{2}}$

$\omega_{2n}^{2k} = g^{\frac{a}{2} \cdot 2k} = g^{ak} = \omega_n^k$

> 2. $\omega_n^0$，$\omega_n^1 \dots$ 互不相同

之前是在一个圆周上取的点，显然不相同。

这里也是不相同的，证明应该要用到剩余系的一些定理，不过如果仅仅是做题不用管那么多。抄一点证明来。带入，发现循环节是 $n$，就可以证明了。

有个小知识点 是kb在modn意义下长啥样

比如3k在mod7意义下

0,3,6,2,5,1,4,0,3,...,

又比如6k在mod10意义下

0,6,2,8,4,0,6,2,...,

你发现这是循环的，而且循环节是n/gcd(n,b)

证明：由于有这个式子

$$\mathrm{ad} \equiv bd (mod\ md) \leftrightarrow \mathrm{a} \equiv b (mod\ m)$$

所以 $jb \equiv kb (mod\ m) \leftrightarrow j\left(\frac{b}{d}\right) \equiv k\left(\frac{b}{d}\right)(mod\,\frac{n}{d})$

但是这个循环具体长啥样是没有任何规律的，你只能知道这个东西就是

$0, \gcd(n,b), 2\gcd(n,b), \dots, n - \gcd(n,b)$按照某种顺序排列

> 3. $-\omega_n^k = \omega_n^{k+\frac{n}{2}}$

$\omega_n^n = g^{an} = g^{(an+1)-1} = g^{p-1} = 1$

$\therefore (\omega_n^{\frac{2}{n}})^2 = 1$

$\therefore \omega_n^{\frac{2}{n}} = \pm 1$

$\because \omega_n^{\frac{2}{n}} \neq \omega_n^0 = 1$

$\therefore \omega_n^{\frac{2}{n}} = -1$

$\therefore \omega_n^k \times \omega_n^{\frac{2}{n}} = -\omega_n^k = \omega_n^{k+\frac{n}{2}}$

证明部分就到这里啦，下面是板子qwq

```
# include <cstdio>
# include <iostream>
# include <cmath>
# define R register int
# define ll long long

using namespace std;

const int maxn=4000006;
const ll g=3;
const ll inv_g=332748118;

const ll mod=998244353;
```

```
int n,m,len,rev[maxn],fg;
ll a[maxn],b[maxn];

ll qui (ll a,ll b)
{
    ll s=1;
    while(b)
    {
        if(b&1LL) s=s*a%mod;
        a=a*a%mod;
        b>>=1LL;
    }
    return s;
}

void NTT (ll *f,int v)
{
    int ln;
    ll og1,og,t;
    for (R i=0;i<len;++i) if(i<=rev[i]) swap(f[i],f[ rev[i] ]);
    for (R i=2;i<=len;i<<=1)
    {
        ln=i>>1;
        og1=qui((v==1)?g:inv_g,(mod-1)/i);
        for (R b=0;b<len;b+=i)
        {
            og=1;
            for (R x=b;x<b+ln;x++)
            {
                t=og*f[ln+x]%mod;
                f[x+ln]=(f[x]-t+mod)%mod;
                f[x]=(f[x]+t)%mod;
                og=og*og1%mod;
            }
        }
    }
}

int read ()
{
    R x=0;
    char c=getchar();
    while (!isdigit(c)) c=getchar();
    while (isdigit(c)) x=(x<<3)+(x<<1)+(c^48),c=getchar();
    return x;
}

int main()
{
    scanf("%d",&n);
    scanf("%d",&m);
    for (R i=0;i<=n;++i) a[i]=read();

    for (R i=0;i<=m;++i) b[i]=read();
```

```
    len=1; while(len<n+m+1) len<<=1;
    for (R i=1;i<len;++i) rev[i]=(rev[i>>1]>>1)|((i&1)?(len>>1):0);
    NTT(a,1);
    NTT(b,1);
    for (R i=0;i<len;++i) a[i]=a[i]*b[i]%mod;
    NTT(a,-1);
    ll inv=qui(len,mod-2);
    for (R i=0;i<=n+m;++i)
        a[i]=a[i]*inv%mod;
    for (R i=0;i<=n+m;++i)
        printf("%lld ",a[i]);
    return 0;
}
```