# 数论变换

看这个之前，要先看懂快速傅里叶变换并且用代码实现它。之前傅里叶变换说到：

F[k]=

=== cos(2πk/n)-i\*sin(2πk/n)

这个Wn计算结果是浮点数，而不是整数，计算机算浮点数比整数慢的多的多，算1亿次整数加减法1秒不到，而浮点数要很久很久。

数论变换是特定情况下，用整数替代W，进行的傅里叶变换

用什么替代呢？

是http://img.blog.csdn.net/20140903144453226 P是一个合适的素数，g代表P的原根

这里为什么我也不知道，只记住结论

原根求法参照另一片文档，

设r= (P-1)/N，

F[k]=

中=

正向变换是：

F[k]=%P

逆变换是：

x[k] =%P

类似FFT

F(k)=E(k)+\*O(k)

正变换中 =g^(k\*(P-1)/N)

那么取P有哪些条件？

要求http://img.blog.csdn.net/20140903152019828是素数且http://img.blog.csdn.net/20140903152349954必须是http://img.blog.csdn.net/20140903152141531的因子。由于http://img.blog.csdn.net/20140903160205078经常是**2**的方幂，所以可以构造形

如http://img.blog.csdn.net/20140903160435203的素数。可以选择http://img.blog.csdn.net/20140903161651206为**费马素数**

由于这个和FFT类似也是处理2的整数次方的数据，因此提前预处理好g^r生成一个数组

有的地方取http://img.blog.csdn.net/20140903161952650，这样P足够大了能应对所有条件，一般（P-1）至少要能整除到2的21次方，也就是200万。

设r= (P-1)/N，

原根求得g=3;r分别是

1004535809,502267904,251133952,125566976,62783488,31391744,15695872,7847936,3923968,1961984,980992,490496,245248,122624,61312,30656,15328,7664,3832,1916,958

那么g^r%P是以下数组

grn[21]={3,1004535808,483363861,395918948,691095095,67253981,89059135,337291080,317143553,8295483,327081633,714163887,295244910,2062645,524615618,333849333,50393163,925609281,615614863,862977694,}

而原本快速傅里叶变换的W(n,k)代表Wn的k次方

它=quickmi(grn[bit[n]],k,P)

其中bit[n]代表n的二进制位数减一

2 4 8 16 32分别对应

1 2 3 4 5

这个P太大会让效率稍微变慢，我找了几个数比较合适的有：

65537=65536+1

786433=3\*262144+1

5767169=11\*524288+1

7340033=7\*1048576+1

998244353=952\*1048576+1

运行make.cpp生成结果

### 编码细节

* P怎么取？太大会稍微降低效率，那我取小的，比如65536，但是如果做多项式乘法，里面逆变换回来的结果要与n\*n复杂度算法结果一致，p太小就给mod掉了65536不行，我取了998244353
* 为了加速g^r用离线打表方式，也就20多个，它的逆元也离线打表
* 多项式相乘应该把不够2的整数次方的N，补0到2的整数次方，且应该统一长度N，

把N取大点不要小于len(a)+lenb(b),比如a多项式有3个系数，b有6个系数，加一起9个那么补0到16，对于a和b的NTT都用16长度，NTT长了不会出，错短了会出错，因为到时候用INTT变回去的时候，len(a\*b)=16，应该统一前后长度

* 如果多个同样长度的N的NTT需要做，那么可以只求一次倒序存下来，这样也有助于节省时间，就比如多项式乘法，的2次NTT和一次INTT，求倒序可以用雷德算法
* 为了减少乘法操作，最内层循环注意用临时变量缓存的手段让乘法只有一次
* FFT的蝶形加速时候最外层循环是for(n=2;n<=N;n\*=2)没错，这里面要首先预处理g^r的从0到n/2次方存下来。
* 接着刚才那个，为什么是n/2次方不是n次方，因为里面计算合并的时候也要有特殊方法只需算一半也就是n/2,而不是n
* 特殊方法是什么呢

来看核心代码temp表示下一层的F数组，F表示当前层的F数组，递推顺序是n由2逐渐翻倍到N

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

for(j=0;j<m;j++){

b=F[i+j];

d=F[i+j+m]\*powg[j]%mod;

temp[i+j]=(b+d)%mod;

temp[i+j+m]=(b-d+mod)%mod;//利用公式少做乘法加速

}

}

其实应该是

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

for(j=0;j<m;j++){

b=F[i+j];

d=F[i+j+m]\*powg[j]%mod;

temp[i+j]=(b+d)%mod;

temp[i+j+m]= (b+powg[m+j]\* F[i+j+m])%mod

=(b+powg[m]\*powg[j]\* F[i+j+m])%mod

=(b+powg[m]\*d)%mod

}

}

以上mod=p

(b+powg[m]\*d)%p中

powg[m]=g^(N/2\*(p-1)/N)%p = g^( (p-1)/2)%p

由原根性质，g^( (p-1)/2)%p=p-1;

(b+powg[m]\*d)%p=(b+(p-1)\*d)%p=(b-d)%p=(b-d+p)%p

NTT常用的素数表：

|  |  |  |  |
| --- | --- | --- | --- |
| r⋅+1 | r | k | g |
| 3 | 1 | 1 | 2 |
| 5 | 1 | 2 | 2 |
| 17 | 1 | 4 | 3 |
| 97 | 3 | 5 | 5 |
| 193 | 3 | 6 | 5 |
| 257 | 1 | 8 | 3 |
| 7681 | 15 | 9 | 17 |
| 12289 | 3 | 12 | 11 |
| 40961 | 5 | 13 | 3 |
| 65537 | 1 | 16 | 3 |
| 786433 | 3 | 18 | 10 |
| 5767169 | 11 | 19 | 3 |
| 7340033 | 7 | 20 | 3 |
| 23068673 | 11 | 21 | 3 |
| 104857601 | 25 | 22 | 3 |
| 167772161 | 5 | 25 | 3 |
| 469762049 | 7 | 26 | 3 |
| 998244353 | 119 | 23 | 3 |
| 1004535809 | 479 | 21 | 3 |
| 2013265921 | 15 | 27 | 31 |
| 2281701377 | 17 | 27 | 3 |
| 3221225473 | 3 | 30 | 5 |
| 75161927681 | 35 | 31 | 3 |
| 77309411329 | 9 | 33 | 7 |
| 206158430209 | 3 | 36 | 22 |
| 2061584302081 | 15 | 37 | 7 |
| 2748779069441 | 5 | 39 | 3 |
| 6597069766657 | 3 | 41 | 5 |
| 39582418599937 | 9 | 42 | 5 |
| 79164837199873 | 9 | 43 | 5 |
| 263882790666241 | 15 | 44 | 7 |
| 1231453023109121 | 35 | 45 | 3 |
| 1337006139375617 | 19 | 46 | 3 |
| 3799912185593857 | 27 | 47 | 5 |
| 4222124650659841 | 15 | 48 | 19 |
| 7881299347898369 | 7 | 50 | 6 |
| 31525197391593473 | 7 | 52 | 3 |
| 180143985094819841 | 5 | 55 | 6 |
| 1945555039024054273 | 27 | 56 | 5 |
| 4179340454199820289 | 29 | 57 | 3 |