# 快速傅里叶变换

转自<http://www.gatevin.moe/acm/fft%E7%AE%97%E6%B3%95%E5%AD%A6%E4%B9%A0%E7%AC%94%E8%AE%B0/>

FFT(快速傅立叶变换)和NTT(快速数论变换)看上去很高端，真正搞懂了就很simple了辣。

首先给出多项式的一些定义(初中数学内容)：

形如Σaixi的式子就是多项式！

多项式中每个单项式叫做多项式的项。

这些单项式中的最高次数，就是这个多项式的次数。

有几个不同的元也是多项式，但在下面将不被考虑。

注意：(n+1)个点可以唯一确定一个n次多项式(两点定线啊之类的)。

然后就是一些比较高明的东西了。

首先在掌握FFT之前我们要掌握一下知识：

1.复数的计算法则.

形如(a+bi)的数叫复数，分为实部和虚部。

i是这么一个东西：i\*i+1=0，虚数单位。

复数的加减法：实部虚部分别相加减。

复数的乘法：(a+bi)\*(c+di)=(ac-bd)+(ad+bc)i；

除法太难打所以请戳[这里](http://baike.baidu.com/link?url=q1bUgtnOP0YfbUPFJlxthrpy5Tp_lps6t1MpUxwjg26nUFRJX_rr-nzbmmHvKSUX98CuGKAXJV5OGorYlF0IIsvTkZsWcp2cwq8M9EQUwRzeuLKsjB4uZFtMGXW1VrO-)。

2.复数的表达形式

~~感谢一位叫~~**~~卜卜~~**~~的热心网友大晚上不看番教我数学。~~

 第一种形式就是代数式：(a+bi)，高中数学内容。

第二种形式~~也许？~~叫三角式：r(cosθ+isinθ)。

具体来说，将代数式里的a,b放到~~二维笛卡尔坐标系~~平面直角坐标系里，横坐标为实部，纵坐标为虚部。把原点和(a,b)相连，记这条向量与X轴的夹角为θ，模长为r，上面那个式子就很好理解了。

那么来看看三角式下的乘法运算？

|  |
| --- |
| r1(cosθ1+isinθ1)\*r2(cosθ2+isinθ2) = r1r2(cos(θ1+θ2)+isin(θ1+θ2)) |

  没错就是这样。

于是就有显而易见的n次方式：

|  |
| --- |
| (r(cosθ+isinθ))^n=r^n(cos(nθ)+sin(nθ)) |

 这在FFT中会用到。还有一个公式是(cosθ+isinθ)=eiθ。推理过程要用到。

 然后是多项式乘法。一个n次的多项式乘上一个m次的多项式，结果是(n+m)次的。

朴素的多项式相乘时间复杂度是O(n^2)的，不够优秀。

而FFT则是利用了**单位复根**的优秀性质来解决了这么一个问题。

首先我们需要把多项式转化成点值表示法，称为**求值**。其逆过程称为**插值**。

这样有一个好处：

两个多项式A,B分别取点(X,Ya)和(X,Yb)，A×B就会取到点(X,Ya\*Yb)；

具体是什么原因？我认为生命需要留下一点遗憾(啧)。

其实很好理解。

T(x)=f(x)\*g(x)，所以T(3)=f(3)\*g(3)。

显而易见。

所以转化成点值表示法后，"相乘"反倒成为最简单的了。

所以多项式相乘的基本步骤：

对A,B求值 » 点值乘法 » 插值。

若能将求值和插值的复杂度降低，就能达到我们的目的了！

FFT的核心思想：

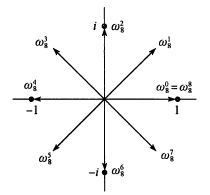
**通过恰当选取x的值，并采用分治策略使得求值和插值的复杂度降下来。**

首先我们要了解的是**n次单位复数根**，记为Wn...Wnn。Wnn= 1 = 1+0\*i；

并且n次单位复数根的个数为n。

**算法导论**告诉我们，nn个单位复数根均匀的分布在以**复平面**的原点为圆心的**单位半径**的圆周上。

~~憋问我为什么~~

(Pic from Xlightgod)

记Wn=r(cosθ+isinθ)。

那么我们可以得知：Wnn=r^n(cos(nθ)+i\*sin(nθ))=(1+0\*i)；

r=1，然后你稍稍推一下就知道θ=0。

设nθ=φ+2kπ，则φ=θ/n+2kπ/n；

因为θ=0，所以就是2kπ/n有值。

所以φ=2kπ/n；

sin和cos都是以2π为周期的，所以可以用φ代替θ

 所以Wn=cosφ+isinφ=e(i\*2kπ/n)。

接下来就可以证明一个重要的定理：

**消去定理**：Wakbk=(e(i\*2kπ/ak))bk=e(i\*2kπ\*bk/ak)=e(i\*2kπ\*b/a)=Wab；

然后用这个定理可以证明：

**折半定理**：(Wnk)2=e(2\*2kπ/n)=e(2kπ/(n/2))=Wn/2k；

这样的话，一次平方下来，取值就少一倍。

接下来就是很简单的 分治 了。

《论折半定理在信息学竞赛中的简单应用》 傅立叶

把这个多项式A(x)=Σaixi分治一下，构建新的多项式。

A[0](x)=a0+a2x+a4x2+...+an-2x(n-2)/2；

A[1](x)=a1+a3x+a5x2+...+an-1x(n-1)/2；

A(x)=A[0](x2)+x\*A[1](x2)；

因为这个是严格分治的，所以最高次项必须要是2的n次方。

(你问我n不是2的幂怎么办？扩大一下，高位系数全为0不就完了

~~所以说常数大得要死。~~

所以我们利用快速傅立叶变换求出了离散傅立叶变换(DFT)。

好像是把求值叫DFT，把插值叫IDFT。

然后又有人证明出插值只要将Wn变成Wn-1，再将结果除以n即可。

再做一遍FFT就可以了。

Congratulations! 复杂度已经被我们降到了O((n+m)log(n+m))。

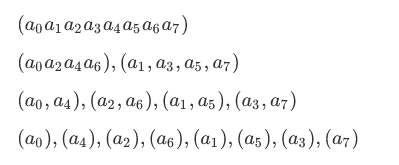
代码实现起来竟然这么短，关键语句只有9行！

//uoj模板题

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | #include    <iostream>  #include    <cstdio>  #include    <cstdlib>  #include    <algorithm>  #include    <vector>  #include    <cstring>  #include    <queue>  #include    <cmath>  #include    <complex>  #define LL long long int  using namespace std;    const int N = 262145;  const double pi = acos(-1.0);  typedef complex<double> dob;  int n,m;  dob a[N],b[N];    int gi()  {    int x=0,res=1;char ch=getchar();    while(ch>'9'||ch<'0'){if(ch=='-')res\*=-1;ch=getchar();}    while(ch<='9'&&ch>='0')x=x\*10+ch-48,ch=getchar();    return x\*res;  }    inline void FFT(dob \*A,int len,int f)  {    if(len==1)return;    dob wn(cos(2.0\*pi/len),sin(f\*2.0\*pi/len)),w(1,0),t;    dob A0[len>>1],A1[len>>1];    for(int i=0;i<(len>>1);++i)A0[i]=A[i<<1],A1[i]=A[i<<1|1];    FFT(A0,len>>1,f);FFT(A1,len>>1,f);    for(int i=0;i<(len>>1);++i,w\*=wn){      t=w\*A1[i];      A[i]=A0[i]+t;      A[i+(len>>1)]=A0[i]-t;    }  }    int main()  {    n=gi();m=gi();    for(int i=0;i<=n;++i)a[i]=gi();    for(int i=0;i<=m;++i)b[i]=gi();    m+=n;    for(n=1;n<=m;n<<=1);    FFT(a,n,1);FFT(b,n,1);    for(int i=0;i<=n;++i)a[i]\*=b[i];    FFT(a,n,-1);    for(int i=0;i<=m;++i)      printf("%d ",int(a[i].real()/n+0.5));    return 0;  } |

然而我们早就知道递归有着巨大的常数，加上FFT的巨大常数(三角函数计算)，导致奇慢无比。

我们来欣赏一下这个美丽的蝴蝶递归。



把最后一行的数化成二进制：

000,100,010,110,001,101,011,111；

然后把每一个数顺序反过来：

000,001,010,011,100,101,110,111；

是个递增的对不对？十分优美对不对？

优美个鬼啊

于是就有人喜(丧)大(心)普(病)奔(狂)推出了三层for人工合并的东西。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #include    <iostream>  #include    <cstdio>  #include    <cstdlib>  #include    <algorithm>  #include    <vector>  #include    <cstring>  #include    <queue>  #include    <cmath>  #include    <complex>  #define LL long long int  using namespace std;    const int N = 262145;  const double pi = acos(-1.0);  typedef complex<double> dob;  int n,m,L,R[N];  dob a[N],b[N];    inline int gi()  {    int x=0,res=1;char ch=getchar();    while(ch>'9'||ch<'0'){if(ch=='-')res\*=-1;ch=getchar();}    while(ch<='9'&&ch>='0')x=x\*10+ch-48,ch=getchar();    return x\*res;  }    inline void FFT(dob \*A,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){      dob wn(cos(pi/i),sin(f\*pi/i)),x,y;      for(int j=0;j<n;j+=(i<<1)){        dob w(1,0);        for(int k=0;k<i;++k,w\*=wn){      x=A[j+k];y=w\*A[j+i+k];      A[j+k]=x+y;      A[j+i+k]=x-y;        }      }    }  }    int main()  {    n=gi();m=gi();    for(int i=0;i<=n;++i)a[i]=gi();    for(int i=0;i<=m;++i)b[i]=gi();    m+=n;    for(n=1;n<=m;n<<=1)++L;    for(int i=0;i<n;++i)R[i]=(R[i>>1]>>1)|((i&1)<<(L-1));    FFT(a,1);FFT(b,1);    for(int i=0;i<=n;++i)a[i]\*=b[i];    FFT(a,-1);    for(int i=0;i<=m;++i)printf("%d ",int(a[i].real()/n+0.5));    return 0;  } |

第一层i是枚举合并到了哪一层。

第二层j是枚举合并区间。

第三层k是枚举区间内的下标。

j\*k=(n+m)；i是log级的。

所以说复杂度没变，常数降下来了。

我们不得不承认FFT是一个优秀而鬼畜的东西。

因为有三角函数和浮点数的参与，FFT有时候会出现尴尬的爆精度现象。

这种病医生说是救不了的。

有些题目要求答案要对一个质数取模，我们知道取模是数论内容。

那么有没有什么东西可以替代单位复根呢？

当然有！原根！

设原根为g。

Wnn≡gP-1≡1(mod P)；

所以可以把g(P-1)/n看成Wn的等价。

好的NTT学完了。

做题时常用的方法：

1答案对小素数取模，且n的数量级在10^5时，可以直接进行FFT，然后将结果加上eps后进行取整操作

2答案对大素数取模，若大素数(约10^9数量级)为二的幂的若干倍+1(kn+1)，那么可以直接进行NTT

如果大素数不是kn+1的形式

那么可以选取3个合适的为kn+1形式的素数，然后分别求解答案

再使用中国剩余定理求得模原大素数的答案

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | #include    <iostream>  #include    <cstdio>  #include    <cstdlib>  #include    <algorithm>  #include    <vector>  #include    <cstring>  #include    <queue>  #include    <complex>  #include    <stack>  #define LL long long int  #define ls (x << 1)  #define rs (x << 1 | 1)  #define MID int mid=(l+r)>>1  using namespace std;    const int N = 300010;  const int Mod = 998244353;  int n,m,L,R[N],g[N],a[N],b[N];    int gi()  {    int x=0,res=1;char ch=getchar();    while(ch>'9'||ch<'0'){if(ch=='-')res\*=-1;ch=getchar();}    while(ch<='9'&&ch>='0')x=x\*10+ch-48,ch=getchar();    return x\*res;  }    inline int QPow(int d,int z)  {    int ans=1;    for(;z;z>>=1,d=1ll\*d\*d%Mod)      if(z&1)ans=1ll\*ans\*d%Mod;    return ans;  }    inline void NTT(int \*A,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(3,(Mod-1)/(i<<1)),x,y;      for(int j=0;j<n;j+=(i<<1)){        int g=1;        for(int k=0;k<i;++k,g=1ll\*g\*gn%Mod){      x=A[j+k];y=1ll\*g\*A[i+j+k]%Mod;      A[j+k]=(x+y)%Mod;A[i+j+k]=(x-y+Mod)%Mod;        }      }    }    if(f==1)return;reverse(A+1,A+n);    int y=QPow(n,Mod-2);    for(int i=0;i<n;++i)A[i]=1ll\*A[i]\*y%Mod;  }    int main()  {    n=gi();m=gi();    for(int i=0;i<=n;++i)a[i]=gi();    for(int i=0;i<=m;++i)b[i]=gi();    m+=n;for(n=1;n<=m;n<<=1)++L;    for(int i=0;i<n;++i)R[i]=(R[i>>1]>>1)|((i&1)<<(L-1));    NTT(a,1);NTT(b,1);    for(int i=0;i<n;++i)a[i]=1ll\*a[i]\*b[i]%Mod;    NTT(a,-1);    for(int i=0;i<=m;++i)printf("%d ",a[i]);    printf("\n");    return 0;  } |

快速傅里叶变换原理还是不太懂（囧）

HDU1402

题目大意：求两个大整数的乘积, 两个大整数长度都不超过50000, 多组数据, 时限1s

这个题就是把整数视作是两个多项式, 每一位就是一项, 那么就相当于是两个最高次数不超过50000的多项式乘积之后在x = 10出的值, 那么这样就很简单了, 直接处理出其多项式A=∑a[i]\*10^i,B=∑b[i]\*10^i然后用FFT计算即可

HDU4609

题意：就是现在给出n个长度不超过100000的树枝 的长度, n <= 100000, 长度都是整数, 求在这n根树枝当中任意取其中3根, 能组成三角形的概率

这个题还是很好的一道计数问题, 首先由于n <= 100000不难想到考虑O(nlogn)的算法, 如果用一个多项式A(x), 其x^k的系数ak表示长度为k的树枝的数量, 那么A(x)的平方当中x^k的系数就是从这些树枝中取两根可重复的和为k的排列数, 稍作处理即可得到A[i]数组其中i表示两根树枝的长度和为i, A[i]表示这样的组合数量是A[i]种

那么对于一个三角形, 考虑其最大的边, 首先将n根树枝排序, 然后从小到大一次考虑其作为组成的三角形中的最长边即可

对于第i根树枝最为最长边, 另外两条边的和需要大于第i根树枝的长度, 也就是之前处理出的A[i]~A[maxLength]的和, 然后去掉用了与第i根之后树枝的组合, 在减去用了第i根树枝作为宁外两个的组合的组数即可, 细节可参见代码注释

UVAlive 4671

题目:定义两个字符串之间的Hamming距离指的是两个相同长度的字符串对应位置字符不同的位置数量, 例如“aab"和”bab"的Hamming距离是1因为第一个字符不一样

现在给出K个两个只包含字符'a', 'b'的字符串A和B, 求A的子串中与B的Hamming距离不超过K的本质不同的子串数量, 即如果"aaa"在A中出现两次及以上且满足条件也只记一次

首先这个题之前以为是可以用后缀自动机做的题然后怎么想都超时了, 后来留着这个坑到现在....

这个题实际上是比较巧妙的用了FFT算法, 对于两个字符串, 用多项式来表示, 因为字符只有‘a'和’b', 用多项式的系数为0或者1表示当前这一个位置是'a'还是'b', 首先如果用1表示‘a', 0表示'b'的话对于两个串A和B怎么判断其有多少位不同呢? 其实可以反过来考虑, 考虑其有多少位都是'a', 多少位都是'b', 我们将串B反过来写, 然后和A做多项式乘法, 那么只有当两者都是’a'(这里记的是'a'为1)时多项式多对应那一项结果的系数会+1, 然后我们在把'b'记做1, 'a'记做0, 再来一次可以得到多少位置都是'b', 这样就能得到对应一段A的子串和B的相同位置个数了, 由于将B反向了再构造的多项式, 和A相乘, 得到的相同位置数会在结果的同一个次幂的项的系数上积累, 所以完成乘法之后根据结果多项式的系数即可知道A哪个对应位置的子串和B有几个相同的字符, 对于字串相同的, Hash判断一下即可

总之这个题就是注意两点, 字符只有'a'和'b', 可以对应为0, 1, 而将B反向, 构造A和B的多项式作乘法能将相同位置的贡献聚集到结果的同一次幂的项上

### [CodeChef COUNTARI Arithmetic Progressions](http://blog.csdn.net/u013738743/article/details/46911323)

题目：给出一个数列A[1~n], 每个数都是不超过30000的正整数, 现在求有多少个三元组(i, j, k)满足 1 <= i < j < k <= n使得 A[i], A[j], A[k]成等差数列 n <= 100000

这个题刚开始想到判断A[i] + A[k] == 2\*A[j]可以用FFT处理出任意两个数的和为T的有多少种, 然而发现这样难处理i, j, k的顺序

后来看了各路题解发现可以分块考虑将整个数列分成K块, 每块 N / K 个连续的元素 (当然不一定都是N / K个, 数量级是这样)

那么考虑三个数的位置

如果三个数再同一块中, 直接对每一块进行枚举后两个数的位置然后查询当前块中需要成为等差数列需要的那个数有多少个, 时间复杂度是O(N/K \* N/K \*K)

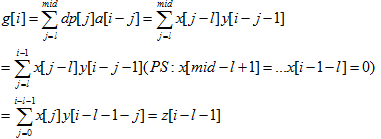
如果三个数中有两个在同一块中, 枚举那两个数, 另外一个数可能在前面的块中也可能在后面的块中, 同上直接查询即可, 时间复杂度是 O(N/ K \* N/K \*K)

如果三个数再不同块中, 枚举中间那个数的在这当前块中的位置N/K种, 然后对于此块前后的部分作FFT得到前后各取一个数和是T有多少种, 对于N/K次直接查询即可, 记最大次数 为M = 1 << 16 (65536), 那么FFT一共进行K次, 时间复杂度是O(K\*MlogM) 综上总体时间复杂度是O(N\*N/K + K\*MlogM), 去一个较为合适的K值即可

HDU 5730

题意：给出长度分别为1~n的珠子，长度为i的珠子有a[i]种，每种珠子有无限个，问用这些珠子串成长度为n的链有多少种方案

题解：令dp[i]表示用这些珠子串成长度为i的链的方案数，令dp[0]=1，轻易得到转移方程  
这里写图片描述

由上式暴力求dp[n]时间复杂度O(n^2)，显然不行，考虑到上式右边是一个卷积形式，所以用CDQ分治+FFT来降低复杂度，假设CDQ(l,r)为求出dp[l],dp[l+1],…,dp[r]的值，那么如果已经通过CDQ(l,mid)求出了dp[l],dp[l+1],…,dp[mid]，下面考虑dp[l],dp[l+1],…,dp[mid]对dp[mid+1],dp[mid+2],…,dp[r]的贡献，令g[i]表示dp[l],…,dp[mid]对dp[i]的贡献，那么有  
这里写图片描述，令x[i]=dp[i+l] (i=0,…,mid-l)，y[i]=a[i+1] (i=0,…,r-l-1)，则有   
   
所以对x序列和y序列做一遍FFT即可得到z序列，进而得到g序列 总时间复杂度O(nlognlogn)