原 十分简明易懂的FFT（快速傅里叶变换）

2018年08月07日 11:38:56 路人黑的纸巾 阅读数 66464

FFT前言

> **快速傅里叶变换 (fast Fourier transform)**,即利用计算机计算离散傅里叶变换（DFT)的高效、快速计算方法的统称，简称FFT。快速傅[illegible]换是1965年由J.W.库利和出的。采用这种算法能使计算机计算离散傅里叶变换所需要的乘法次数大为减少，特别是被变换的抽样点数N越多，FFT算法计算量的节[illegible]显著。
> **FFT（Fast Fourier Transformation）** 是离散傅氏变换（DFT）的快速算法。即为快速傅氏变换。它是根据离散傅氏变换的奇、偶、虚、实等特性，对离散傅立叶变换改进获得的。
> ——百度百科

FFT（Fast Fourier Transformation），中文名**快速傅里叶变换**，用来**加速多项式乘法**
朴素高精度乘法时间$O(n^2)$，但FFT能$O(n\log_2 n)$的时间解决
FFT名字逼格高，也难懂，其他教程写得让人看不太懂，于是自己随便写一下

> - 建议对**复数**、**三角函数**相关知识有所耳闻 ~~（不会也无所谓）~~
> - 下面难懂的点我会从网上盗

多项式的系数表示法和点值表示法

- FFT其实是一个用$O(n\log_2 n)$的时间将一个用**系数表示**的多项式转换成它的**点值表示**的算法
- 多项式的系数表示和点值表示可以**互相转换**

系数表示法

一个**n-1次n项**多项式$f(x)$可以表示为$f(x)=\sum_{i=0}^{n-1}a_ix^i$
也可以用**每一项的系数**来表示$f(x)$，即

$$f(x)=\{a_0,a_1,a_2,...,a_{n-1}\}$$

这就是**系数表示法**，也就是平时~~数学课上~~用的方法

点值表示法

- 把多项式放到平面直角坐标系里面，看成一个**函数**
- 把n个不同的x代入，会得出n个不同的y，在坐标系内就是n个不同的点
- 那么这n个点**唯一确定**该多项式，也就是**有且仅有**一个多项式满足$\forall k,f(x_k)=y_k$
- 理由很简单，把n条式子联立起来成为一个**有n条方程的n元**方程组，每一项的系数都可以解出来

那么$f(x)$还可以用$f(x)=\{(x_0,f(x_0)),(x_1,f(x_1)),(x_2,f(x_2)),...,(x_{n-1},f(x_{n-1}))\}$来表示
这就是**点值表示法**

高精度乘法下两种多项式表示法的区别

对于两个用**系数表示**的多项式，我们把它们相乘
设两个多项式分别为$A(x),B(x)$
我们要枚举A每一位的系数与B每一位的系数相乘
那么系数表示法做多项式乘法**时间复杂度**$O(n^2)$

但两个用**点值表示**的多项式相乘，只需要$O(n)$的时间

> **什么意思呢?**

设两个点值多项式分别为

$$f(x)=\{(x_0,f(x_0)),(x_1,f(x_1)),(x_2,f(x_2)),...,(x_{n-1},f(x_{n-1}))\}$$
$$g(x)=\{(x_0,g(x_0)),(x_1,g(x_1)),(x_2,g(x_2)),...,(x_{n-1},g(x_{n-1}))\}$$

设它们的乘积是$h(x)$，那么

$$h(x)=\{(x_0,f(x_0)\cdot g(x_0)),(x_1,f(x_1)\cdot g(x_1)),...,(x_{n-1},f(x_{n-1})\cdot g(x_{n-1}))\}$$

所以这里的时间复杂度只有一个枚举的$O(n)$

- 突然感觉高精度乘法能$O(n)$**暴艹**一堆题?
- 但是**朴素**的系数表示法转点值表示法的算法还是$O(n^2)$的，逆操作类似
- **朴素**系数转点值的算法叫**DFT（离散傅里叶变换）**，点值转系数叫**IDFT（离散傅里叶逆变换）**
- 难道高精度乘法只能$O(n^2)$了吗?

DFT前置知识&技能

复数

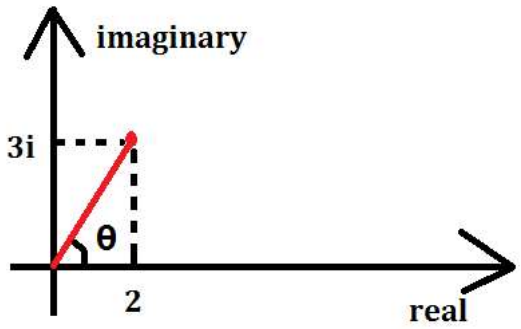
毕竟高中有所以不多说

> 我们把形如a+bi（a,b均为实数）的数称为**复数**，其中a称为实部，b称为虚部，i称为虚数单位。当虚部等于零时，这个复数可以视为实数；当z的虚部不等于零时，实部常称z为纯虚数。复数域是实数域的代数闭包，也即任何复系数多项式在复数域中总有根。 复数是由意大利米兰学者卡当在十六世纪首次引入，经过达朗贝尔、棣莫弗、等人的工作，此概念逐渐为数学家所接受。
> ——百度百科

初中数学老师会告诉你没有$\sqrt{-1}$，但仅限R
扩展至复数集C，定义$i^2=-1$，一个复数z可以表示为$z=a+bi(a,b\in R)$
其中a称为**实部**，b称为**虚部**，i称为**虚数单位**

- 在复数集中就可以用i表示负数的平方根，如$\sqrt{-7}=\sqrt{7}i$

还可以把复数看成**复**平面直角坐标系上的**一个点**，比如下面



x轴就是实数集中的坐标轴，y轴就是虚数单位i轴

这个点$(2,3)$表示的**复数**就是$2+3i$，或者想象它代表的**向量**为$(2,3)$
其实我们还可以自己想象 ~~（其实没有这种表达方式）~~ 它可以表示为$(\sqrt{13},\theta)$
一个复数z的**模**定义为它到原点的距离，记为$|z|=\sqrt{a^2+b^2}$
一个复数$z=a+bi$的**共轭复数**为$a-bi$（虚部取反），记为$\overline{z}=a-bi$

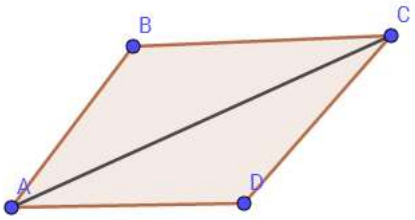
复数的运算

复数不像点或向量，它和实数一样可以进行**四则运算**
设两个复数分别为$z_1=a+bi, z_2=c+di$，那么

$$z_1+z_2=(a+c)+(b+d)i$$

$$z_1z_2=(ac-bd)+(ad+bc)i$$

复数**相加**也满足**平行四边形法则**



这张是从网上盗的

即$AB+AD=AC$

复数**相乘**还有一个值得注意的**小性质**

$$(a_1,\theta_1)*(a_2,\theta_2)=(a_1a_2,\theta_1+\theta_2)$$

即**模长相乘，极角相加**

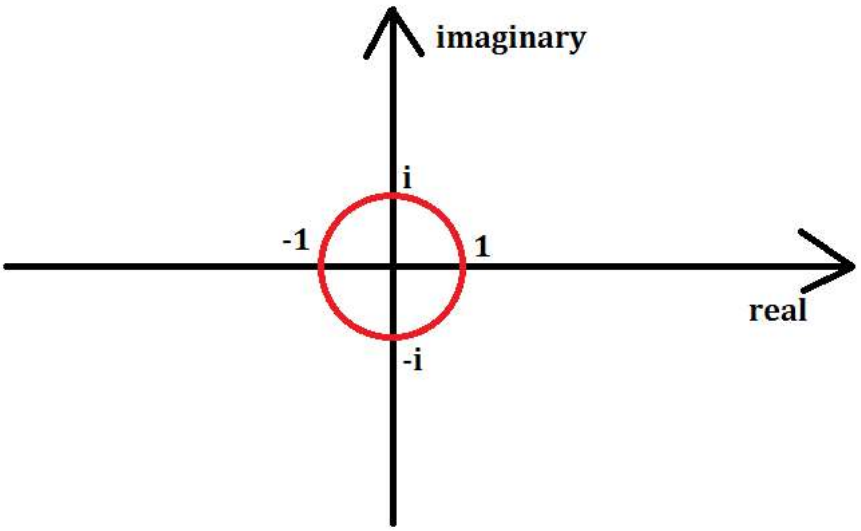
DFT（离散傅里叶变换）

- **一定注意从这里开始所有的n都默认为2的整数次幂**

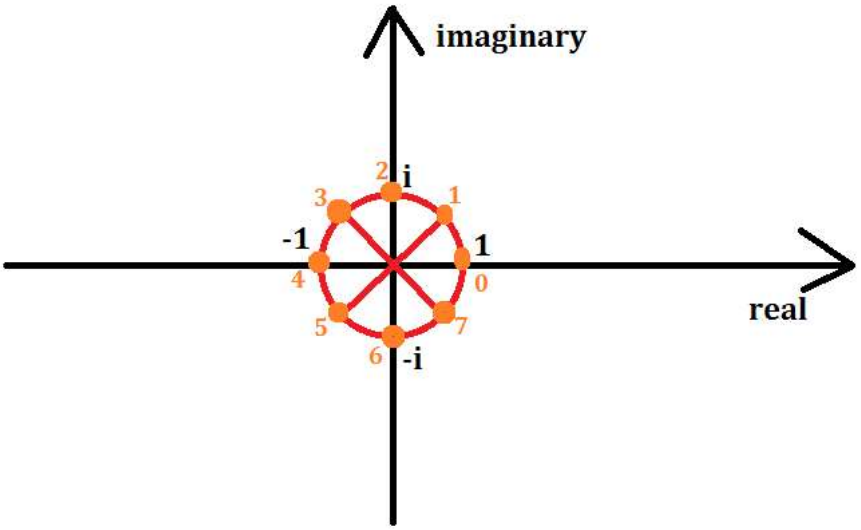
对于任意系数多项式转点值，当然可以**随便**取任意n个x值代入计算
但是**暴力**计算$x_k^0, x_k^1, ..., x_k^{n-1} (k \in [0, n))$当然是$O(n^2)$的时间
其实可以代入一组**神奇**的x，代入以后不用做那么多的次方运算
这些x当然不是乱取的，而且取这些x值应该就是 **傅里叶** 的主意了

考虑一下，如果我们代入一些x，使每个x的**若干次方**等于1，我们就不用做全部的次方运算了
± 1是可以的，考虑虚数的话$\pm i$也可以，但只有这四个数远远不够

- 傅里叶说：**这个圆圈上面的点都可以做到**



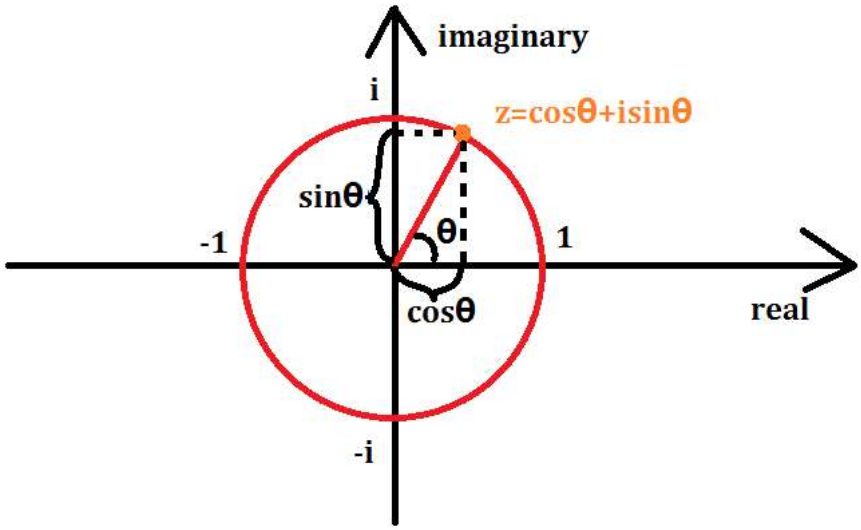
以原点为圆心，画一个半径为1的**单位圆**
那么单位圆上所有的点都可以经过若干次次方得到1
傅里叶说还要把它给n等分了，比如此时$n = 8$



橙色点即为$n=8$时要取的点，从$(1,0)$点开始，逆时针从0号开始标号，标到7号
记编号为k的点代表的复数值为ω_n^k，那么由**模长相乘，极角相加**可知$(\omega_n^1)^k=\omega_n^k$
其中ω_n^1称为n**次单位根**，而且每一个ω都可以求出 ~~（我三角函数不好）~~

$$\omega_n^k=\cos\frac{k}{n}2\pi+i\sin\frac{k}{n}2\pi$$

或者说也可以这样解释这条式子



注意$sin^2\theta+cos^2\theta=1$什么的，就容易理解了

那么$\omega_n^0,\omega_n^1,...,\omega_n^{n-1}$即为我们要代入的$x_0,x_1,...,x_{n-1}$

单位根的一些性质

推FFT的过程中需要用到ω的一些性质

$$\omega_n^k=\omega_{2n}^{2k}$$

- 它们表示的点（或向量）表示的复数**是相同的**
- **证明**
- $\omega_n^k=cos\frac{k}{n}2\pi+isin\frac{k}{n}2\pi=cos\frac{2k}{2n}2\pi+isin\frac{2k}{2n}2\pi=\omega_{2n}^{2k}$

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

- 它们表示的点**关于原点对称**，所表示的复数**实部相反**，所表示的向量**等大反向**
- **证明**
- $\omega_n^{\frac{n}{2}}=cos\frac{\frac{n}{2}}{n}2\pi+isin\frac{\frac{n}{2}}{n}2\pi=cos\pi+isin\pi=-1$
- （这个东西和$e^{ix}=cosx+isinx$与$e^{i\pi}+1=0$有点关系，~~我不会就不讲了~~）

> $\omega_n^0 = \omega_n^n$

- 都等于1，或$1+0i$

FFT（快速傅里叶变换）

虽然DFT搞出来一堆**很牛逼**的ω作为代入多项式的x值
但只是代入这类特殊x值法的变换叫做DFT而已，还是要代入单位根暴力计算

- **DFT还是暴力$O(n^2)$...**

但DFT可以**分治**来做，于是 **FFT（快速傅里叶变换）** 就出来了
首先设一个多项式$A(x)$

$$A(x)=\sum_{i=0}^{n-1}a_ix^i=a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}$$

按$A(x)$下标的**奇偶性**把$A(x)$分成两半，右边再提一个x

$$A(x)=(a_0+a_2x^2+...+a_{n-2}x^{n-2})+(a_1x+a_3x^3+...+a_{n-1}x^{n-1})$$
$$=(a_0+a_2x^2+...+a_{n-2}x^{n-2})+x(a_1+a_3x^2+...+a_{n-1}x^{n-2})$$

两边好像非常相似，于是再设两个多项式$A_1(x),A_2(x)$，令

$$A_1(x)=a_0+a_2x+a_4x^2+...+a_{n-2}x^{\frac{n}{2}-1}$$
$$A_2(x)=a_1+a_3x+a_5x^2+...+a_{n-1}x^{\frac{n}{2}-1}$$

比较明显得出

$$A(x)=A_1(x^2)+xA_2(x^2)$$

再设$k<\frac{n}{2}$，把ω_n^k作为x代入$A(x)$~~（接下来几步变换要多想想单位根的性质）~~

$$A(\omega_n^k)=A_1((\omega_n^k)^2)+\omega_n^kA_2((\omega_n^k)^2)$$
$$=A_1(\omega_n^{2k})+\omega_n^kA_2(\omega_n^{2k})=A_1(\omega_{\frac{n}{2}}^k)+\omega_n^kA_2(\omega_{\frac{n}{2}}^k)$$

那么对于$A(\omega_n^{k+\frac{n}{2}})$的话，代入$\omega_n^{k+\frac{n}{2}}$

$$A(\omega_n^{k+\frac{n}{2}})=A_1(\omega_n^{2k+n})+\omega_n^{k+\frac{n}{2}}A_2(\omega_n^{2k+n})$$
$$=A_1(\omega_n^{2k}\omega_n^n)-\omega_n^kA_2(\omega_n^{2k}\omega_n^n)$$
$$=A_1(\omega_n^{2k})-\omega_n^kA_2(\omega_n^{2k})=A_1(\omega_{\frac{n}{2}}^k)-\omega_n^kA_2(\omega_{\frac{n}{2}}^k)$$

- **发现了什么?**

$A(\omega_n^k)$和$A(\omega_n^{k+\frac{n}{2}})$两个多项式**后面一坨东西只有符号不同**
就是说，如果已知$A_1(\omega_{\frac{n}{2}}^k)$和$A_2(\omega_{\frac{n}{2}}^k)$的值，我们就可以**同时**知道$A(\omega_n^k)$和$A(\omega_n^{k+\frac{n}{2}})$的值
现在我们就可以递归**分治**来搞FFT了

每一次回溯时只扫当前前面一半的序列，即可得出后面一半序列的答案
$n==1$时只有一个常数项，直接$return$
时间复杂度$O(n\log_2 n)$

IFFT（快速傅里叶逆变换）

想一下，我们不仅要会FFT，还要会**IFFT（快速傅里叶逆变换）**
我们把两个多项式相乘 **（也叫求卷积）**，做完两遍FFT也知道了积的多项式的**点值表示**
可我们平时用系数表示的多项式，点值表示没有意义

- 怎么把点值表示的多项式**快速**转回系数表示法?

- $IDFT$暴力$O(n^2)$做？其实也可以用FFT用$O(n\log_2 n)$的时间搞

你有没有想过为什么傅里叶是把ω_n^k作为x代入**而不是别的什么数？**
原因是有的~~但是有我也看不懂~~
由于我是沙雕所以只用记住**一个结论**

- **一个多项式在分治的过程中乘上单位根的共轭复数，分治完的每一项除以n即为原多项式的每一项系数**

意思就是说FFT和$IFFT$可以**一起搞**

朴素版FFT板子

$c++$有自带的复数模板$complex$库
$a.real()$即表示复数a的实部

```
#include<complex>
#define cp complex<double>

void fft(cp *a,int n,int inv)//inv是取共轭复数的符号
{
    if (n==1)return;
    int mid=n/2;
    static cp b[MAXN];
    fo(i,0,mid-1)b[i]=a[i*2],b[i+mid]=a[i*2+1];
    fo(i,0,n-1)a[i]=b[i];
    fft(a,mid,inv),fft(a+mid,mid,inv);//分治
    fo(i,0,mid-1)
    {
        cp x(cos(2*pi*i/n),inv*sin(2*pi*i/n));//inv取决是否取共轭复数
        b[i]=a[i]+x*a[i+mid],b[i+mid]=a[i]-x*a[i+mid];
    }
    fo(i,0,n-1)a[i]=b[i];
}
```

两个多项式a,b相乘再转系数多项式c，通常只用打这么一小段

```
    cp a[MAXN],b[MAXN];
    int c[MAXN];
    fft(a,n,1),fft(b,n,1);//1系数转点值
    fo(i,0,n-1)a[i]*=b[i];
    fft(a,n,-1);//-1点值转系数
    fo(i,0,n-1)c[i]=(int)(a[i].real()/n+0.5);//注意精度
```

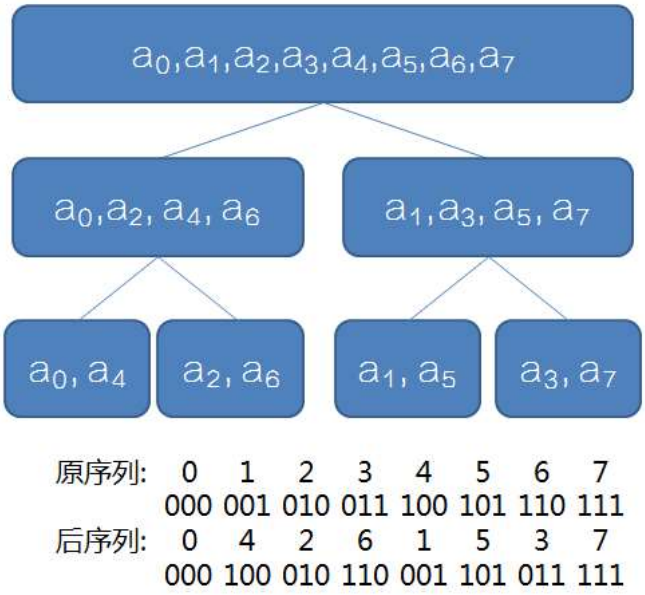
很明显，**FFT只能处理n为2的整数次幂的多项式**
所以在FFT前一定要把n调到2的次幂

这个板子看着**好像**很优美，但是

递归**常数太大**，要考虑优化...

FFTの优化——迭代版FFT

~~这个图也是盗的~~



这个很容易发现点什么吧?

- **每个位置分治后的最终位置为其二进制翻转后得到的位置**

这样的话我们可以先把原序列变换好，把每个数放在最终的位置上，再一步一步向上合并
一句话就可以$O(n)$预处理第i位最终的位置$rev[i]$

```
fo(i,0,n-1)rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
```

~~至于蝴蝶变换**它死了**其实是我不会~~

真•FFT板子

```
void fft(cp *a,int n,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]]);//不加这条if会交换两次（就是没交换）
    }
    for (int mid=1;mid<n;mid*=2)//mid是准备合并序列的长度的二分之一
    {
        cp temp(cos(pi/mid),inv*sin(pi/mid));//单位根，pi的系数2已经约掉了
        for (int i=0;i<n;i+=mid*2)//mid*2是准备合并序列的长度，i是合并到了哪一位
        {
            cp omega(1,0);
            for (int j=0;j<mid;j++,omega*=temp)//只扫左半部分，得到右半部分的答案
            {
                cp x=a[i+j],y=omega*a[i+j+mid];
                a[i+j]=x+y,a[i+j+mid]=x-y;//这个就是蝴蝶变换什么的
            }
```

```
        }
    }
}
```


~~这个板子好像不是那么好背~~
至少这个板子已经**很优美**了

FFT后记

~~本人版权意识薄弱……~~

> - **本博客部分知识学习于**
> - **https://www.cnblogs.com/RabbitHu/p/FFT.html**
> - **https://www.cnblogs.com/zwfymqz/p/8244902.html?mType=Group#_label3**
> - **https://blog.csdn.net/ggn_2015/article/details/68922404**

NTT我来了