快速傅里叶变换(Fast Fourier Transform)学习总结

Voiphon

1、引言

这玩意儿很久以前就在算导上看过，然而当时并没有看懂。这两天想起来去网上翻，然而网上并没有比较简单的适合我这等蒟蒻看的资料（全 TM 是数字信号分析，……好吧其实还是有两个 OIer 写的不错的[1]）。于是我就打算整理一下，不然过几天又要忘掉了……

2、卷积（多项式乘法）

定义一个 n 次多项式 $f(x)=a_0+a_1x+\cdots+a_{n-1}x^{n-1}$（这儿的次数有点不正常别理它）。

然后两个多项式 $f(x)$ 与 $g(x)$ 的乘积显然还是一个多项式。我们管这个乘积叫多项式 $f(x)$ 与 $g(x)$ 的卷积。（其实真正的卷积是定义在连续函数上的积分……）

怎么算？整式乘法……

即 $h(x)=f(x)\cdot g(x)=a_0b_0+(a_0b_1+a_1b_0)x+\cdots+\sum_{i=0}^{k}a_ib_{k-i}x^k+\cdots$

把那个系数提取出来 $c_k=\sum_{i=0}^{k}a_ib_{k-i}$ ，这就是两个数列的卷积了（也就是实际比赛里面要算的东西……）

朴素计算方法：直接代上面那个式子，时间复杂度 O(n^2)

3、多项式的点值表达、求值与插值

一个 n 次的多项式除了能用那一坨系数 $(a_0,a_1,\cdots,a_{n-1})$ 表达，还能用 n 个互不重合的点 $(x_0,y_0),(x_1,y_1),\cdots,(x_{n-1},y_{n-1})$ 来表达（因为一共就 n 个常数）。

然后点值表达的好处是什么？就是点值表达计算卷积非常快……

也就是把 x 相同的点的函数值直接相乘就好了，时间复杂度 O(n)

从系数表达变换为点值表达的过程叫求值。

怎么做？直接代入 n 个点计算，然后每一次计算长这样：$f(x)=a_0+x(a_1+x(a_2+\cdots$

或者直接循环里面算一次把 x 乘一下也行，反正单个点计算时间 O(n)，总时间 O(n^2)

从点值表达变换为系数表达的过程叫插值。

怎么做？~~待定系数法~~。拉格朗日插值法：这篇文章用不到，自行百度。

4、DFT(Discrete Fourier Transform) 离散傅里叶变换 O(n^2)

我刚刚好像没说我的多项式的 x 必须是实数吧！

所谓这个看上去很高大上的 DFT，其实就是对一个多项式在 n 次单位根上求值。

[1] ①http://blog.csdn.net/zxn0803/article/details/51361111 ②http://blog.csdn.net/iamzky/article/details/22712347

n 次单位根是什么？就是这样的 n 个复数：$\omega_n^k = e^{2\pi ki/n} = \cos\left(k \cdot \dfrac{2\pi}{n}\right) + i\sin\left(k \cdot \dfrac{2\pi}{n}\right)$

通俗说来，就是单位圆以(1,0)为一个顶点的内接正 n 边形的 n 个顶点的坐标。

然后计算就好了……

不过还是甩个公式：$DFT(k) = y_k = f(\omega_n^k) = \sum_{i=0}^{n-1} a_i (\omega_n^k)^i$

5、FFT 快速傅里叶变换 O(nlogn)

这是正文！这是正文！这是正文！重要的事情说三遍！

FFT 是一种能在 O(nlogn)时间内计算 DFT 的神奇的算法。

它的神奇之处在于利用了 n 次单位根的一些玄学性质，然后利用分治大大加速计算。

首先假设 n 为偶数，则有 $(\omega_n^k)^2 = (\omega_n^{k+n/2})^2 = \omega_{n/2}^k$ （三角函数验证即可……）

然后观察多项式：$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^{n-1}$

把它拆成这样子：$f_1(x) = a_0 + a_2 x^2 + a_4 (x^2)^2 + \cdots + a_{n-2} (x^2)^{n/2-1}$

$f_2(x) = a_1 + a_3 x^2 + a_5 (x^2)^2 + \cdots + a_{n-1} (x^2)^{n/2-1}$

显然有 $f(x) = f_1(x) + x \cdot f_2(x)$

看到那个闪亮亮的 x^2 没有？换成其它的 n 个 x 求值，x^2 依然有 n 个；然而在 n 次单位根上求值时，x^2 仅仅只有 n/2 个！

所以 $f_1(x)$ 和 $f_2(x)$ 总计也只要对 n 个点求值（其它 x 的话总计要对 2n 个点求值）

然后如果你的 n 是 2 的幂，一路分治下去，就达到了 O(nlogn)的时间复杂度了……

如果你的 n 不是 2 的幂怎么办？很简单，多项式前面的系数补 0 嘛……

代码最后给……

6、IDFT(Inverse Discrete Fourier Transform) 离散傅里叶逆变换

DFT 是用来求值的，那么既然是逆变换，这货就是用来插值的。

【以下涉及到线性代数的内容高能如有不适请直接看结论】

把 DFT 写成矩阵形式：

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & (\omega_n^1)^2 & \cdots & (\omega_n^1)^{n-1} \\ 1 & \omega_n^2 & (\omega_n^2)^2 & \cdots & (\omega_n^1)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & (\omega_n^{n-1})^2 & \cdots & (\omega_n^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

然后观察左边那个矩阵是范德蒙德矩阵(Vn)，这货的逆矩阵是前人帮我们求好了的，因此我们直接用就可以了。（顺便说一下：这货的行列式的值也是前人求好的，由此可以暴推

拉格朗日插值法……）

于是我们利用范德蒙德逆矩阵~~厚颜无耻~~十分轻松地推导出了 IDFT 的公式：

$$IDFT(k)=a_k=\frac{1}{n}\sum_{i=0}^{n-1}y_i\left(\omega_n^{-k}\right)^i$$

等等，似乎有什么不对的地方——

$$DFT(k)=y_k=\sum_{i=0}^{n-1}a_i(\omega_n^k)^i$$

这俩式子为什么长那么像[2]？【邪恶】那岂不是我们直接把 FFT 的参数改一下套上去就能完成 O(nlogn)时间的插值了？

于是——

7、回到卷积

我们发现可以这样来算卷积：$f\cdot g=IDFT(DFT(f)\cdot DFT(g))$

也就是先求值把两多项式变成点值表达，卷积之后再插值成新多项式。

利用 FFT 优化，总时间 O(nlogn)+O(n)+O(nlogn)=O(nlogn)！

于是我们有了卷积的快速算法——FFT！

代码 Pascal 大约 25 行……C++只要 14 行……

拓展：FFT 涉及到浮点运算常数略大，有没有时间复杂度相同，但常数上更快的算法呢？

答案：NTT（快速数论变换），非常 BT 的东西然而我不会写……

8、应用

CodeVS 3123 Super A*B Problem（模板题）

给出 A、B，求 A*B。其中 A,B 不超过 10W 位。

分析：高精度数字就是 x=10 的多项式……或者直接看成卷积也可以……

代码：@Voiphon 即得（其实下面那个就是核心）

9、递归代码模板（与归并排序相似）【我认为你们分得清 L 和 1 的】

```
procedure FFT(var a:hugeint;l,r,dir:longint);
    var
        w,wn,t:complex;
        n,i,j,k:longint;
    begin
        n:=r-l+1;
        if n=1 then exit;
        wn.Re:=cos(dir*2*pi/n);
        wn.Im:=sin(dir*2*pi/n);
        w.Re:=1;
        w.Im:=0;
        for i:=0 to n shr 1-1 do begin
            tmp[l+i]:=a[l+i*2];
            tmp[l+i+n shr 1]:=a[l+i*2+1];
        end;
        for i:=l to r do a[i]:=tmp[i];
        FFT(a,l,l+n shr 1-1,dir);
        FFT(a,l+n shr 1,r,dir);
        for i:=0 to n shr 1-1 do begin
            t:=w*a[l+i+n shr 1];
            tmp[l+i]:=a[l+i]+t;
            tmp[l+i+n shr 1]:=a[l+i]-t;
            w:=w*wn;
        end;
        for i:=l to r do a[i]:=tmp[i];
    end;
```

[2] 这不是偶然而是必然，事实上 DFT 和 IDFT 是复平面上两个函数域的变换规则，如镜像一样对称。

10、非递归代码模板

没写……利用二进制反转，优化常数而且写起来更简单……自行百度（其实我第一页安利的那两篇博客中就有）……