FFT算法的理解及其代码实现

<https://blog.csdn.net/xzxxzx401/article/details/72861038>

# 多项式乘法问题引入

计算两个**n阶多项式的加法，需要Θ(n)的时间**，只需要将两式对应次数的系数相加即可。**但是两个n阶多项式A、B相乘需要Θ(n2)的时间，因为我们需要将A式的每一个系数乘上B式中每一个系数，**再找次数相同的系数进行合并。我们希望加速多项式相乘的过程。

多项式有两种表示法，**系数表示法**与**点值表示法**。

### 系数表示法表示一个多项式A可以记为：



### 点值法表示多项式B可以记为：



可以证明，一个包含n个点的**点值表示法**可以唯一确定一个n-1次多项式(最高项次数为n-1)。

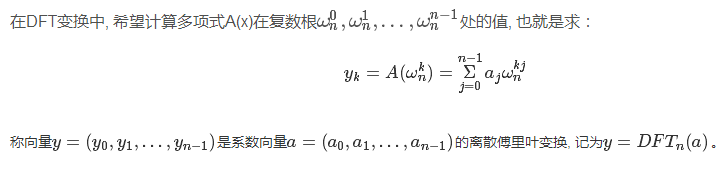
点值表示法得优势在于：对于点值表示的多项式，相加相乘都很方便，我们**只需要将同一x对应的y相乘，就得到新多项式的点值表示**。注意，因为两个n次多项式的乘积是2n次，所以我们在选点时要选2n个点。这样乘起来才会得到2n个点。所以我们**快速做多项式乘法的思路就是先将系数表示的多项式转化成点值表示，相乘，在转化回系数表示**。

但是一般而言，对于系数表示法的多项式A，我们要求出2n个点，需要Θ(n2)的时间。因为我们求一个x对应的A(x)需要Θ(n)的时间。



但是如果我们选取2n次单位复数根，并利用**单位复数根单位性质**进行分治，就可以将复杂度降至Θ(nlogn)。

# 算法分析

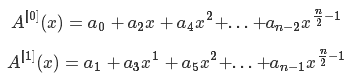


FFT：

直接计算DFT的复杂度仍是Θ(n2)，但是我们利用**消去引理**的一个推论：

，结合分治的策略，可以将复杂度降为Θ(nlogn)。

我们假定n为2的整次幂，若不是，在前面填0补足即可。每一步将当前的多项式A(x), 次数是2的倍数, 分成两个部分：



A[0](x)代表偶数次幂项；A[1](x)代表奇数次幂项。

于是就有了：



用x2换元x，就可以得到原来的n阶多项式。

那么我们如果能求出次数界是n/2的多项式A[0](x)和A[1](x)在n个n次单位复数根的平方处的取值就可以了, 即在：

处的值。

折半原理：

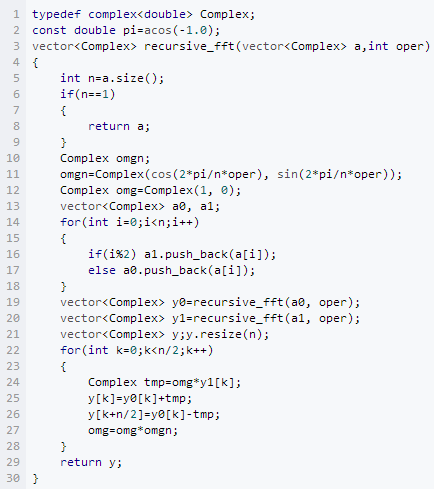


根据折半引理，这n个数其实只有n/2个不同的值，因为相反数平方后相等。也就是说，对于每次分出的两个次数界n/2的多项式, 只需要求出其n/2个不同的值即可，那么问题就递归到了原来规模的一半。

我们根据这个式子可以得到一个递归的实现。

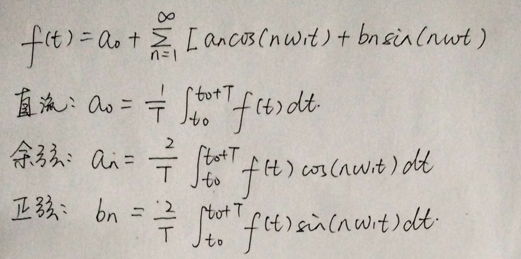
# FFT算法的实现

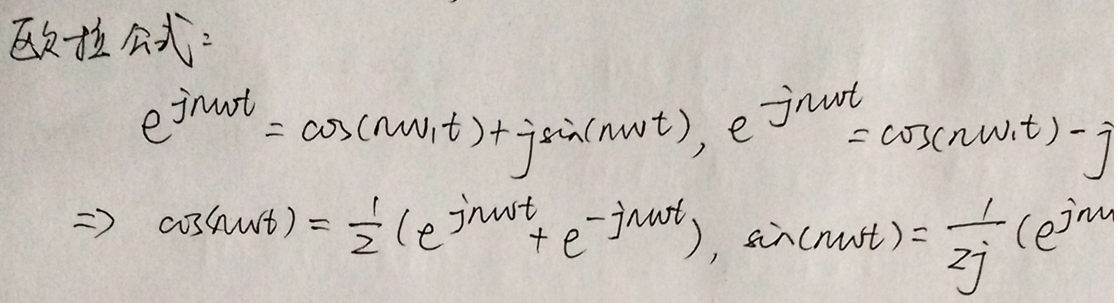
递归版本的FFT实现：

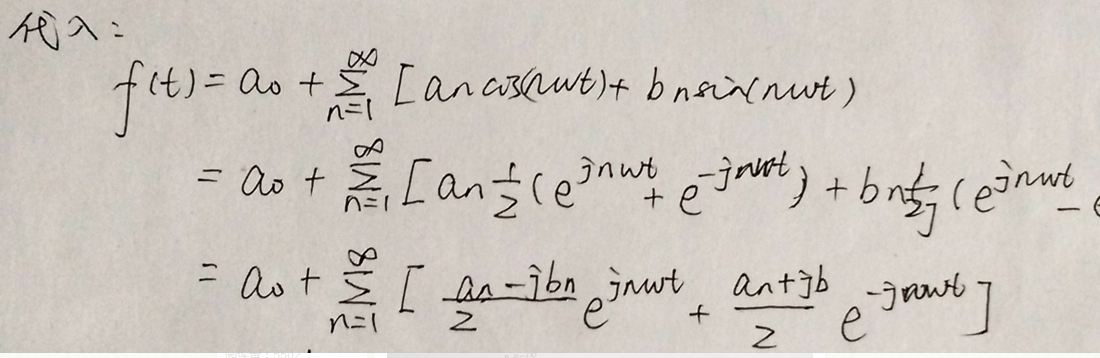


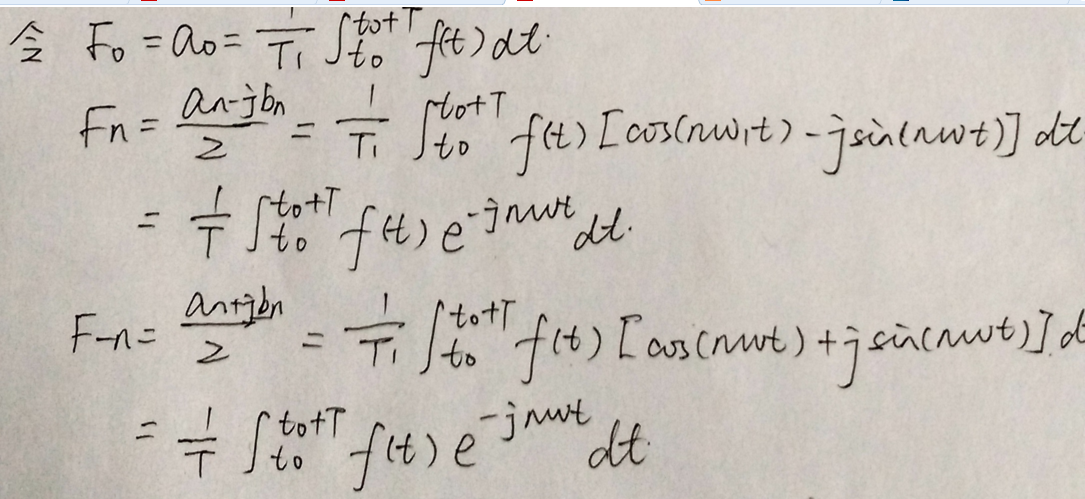
调用之前保证vector a的size已经被置为2的整数次幂。

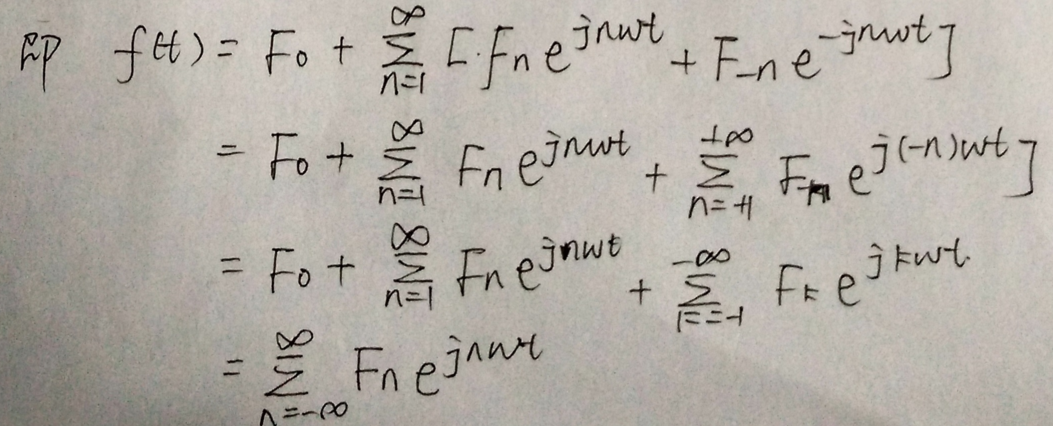
# 从傅里叶级数到傅里叶变换



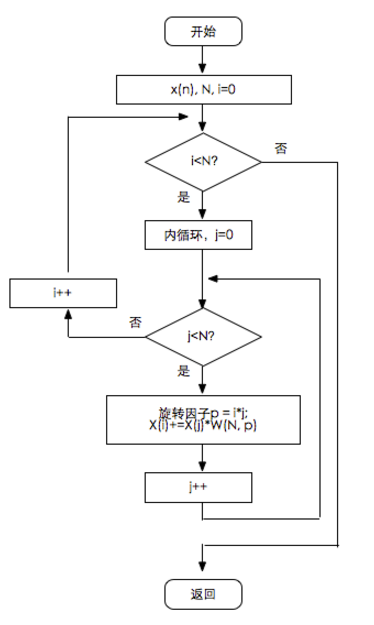




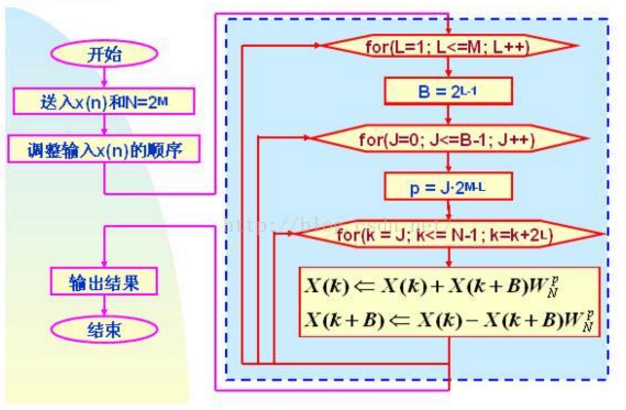




# DFT流程图和代码如下



# 编程思路流程图



#include <iostream>

#include <cstdio>

#include <cmath>

#include <algorithm>

#define PI 3.1415926 //定义圆周率值

#define N 16 //变换点数

using namespace std;

typedef struct compx

{

float real;

float img;

}compx;

//复数乘法

compx Multi(compx a, compx b)

{

compx c;

c.real = a.real\*b.real - a.img\*b.img;

c.img = a.real\*b.img + a.img\*b.real;

return c;

}

//复数加法

compx Add(compx a, compx b)

{

compx c;

c.real = a.real + b.real;

c.img = a.img + b.img;

return c;

}

//复数减法

compx Sub(compx a, compx b)

{

compx c;

c.real = a.real - b.real;

c.img = a.img - b.img;

return c;

}

//快速傅里叶

void FFT(compx data[])

{

int k, i, j = 0;

compx u, w, t;

//自然顺序变成倒位序

for(i = 0; i < N-1; i++)

{

if(i < j)

{

t.real = data[i].real;

t.img = data[i].img;

data[i].real = data[j].real;

data[i].img = data[j].img;

data[j].real = t.real;

data[j].img = t.img;

}

k = N/2;

//实现进位

while(k <= j) //左边最高位为1

{

j = j-k; //最高位1变为0

k /= 2;

}

j += k;

}

//计算蝶形级数m

int f = N, m;

for(m = 1; (f=f/2) != 1; m++);

//控制级数

for(int n = 1; n <= m; n++)

{

int a = 2<<(n-1); //相邻蝶形结的距离.2的n次方

int b = a/2; //相邻运算点的距离

//printf("a = %d b = %d\n", a,b);

u.real = 1.0; //蝶形运算的系数

u.img = 0.0;

w.real = cos(PI/b);//当前系数与上一系数的商

w.img = -sin(PI/b);

int x1, x2; //参加蝶形运算的两个节点

for(j = 0; j < b; j++)

{

//printf("b = %d\t%lf\t%lf\n",b, u.real, u.img);

for(x1 = j; x1 < N; x1 += a)

{

x2 = x1 + b;

t = Multi(data[x2], u);

data[x2] = Sub(data[x1], t);

data[x1] = Add(data[x1], t);

//printf("x1 = %d, x2 = %d\n", x1, x2);

}

u = Multi(u, w);

}

}

}

int main()

{

compx data[N];

for(int i = 0; i < N; i++)

{

data[i].real = i;

data[i].img = 0;

}

**FFT(data);**

for(int i = 0; i < N; i++)

printf("%lf \t%lf\n", data[i].real, data[i].img);

return 0;

}