再探快速傅里叶变换

毛啸

长沙市雅礼中学

April 28, 2016

Contents

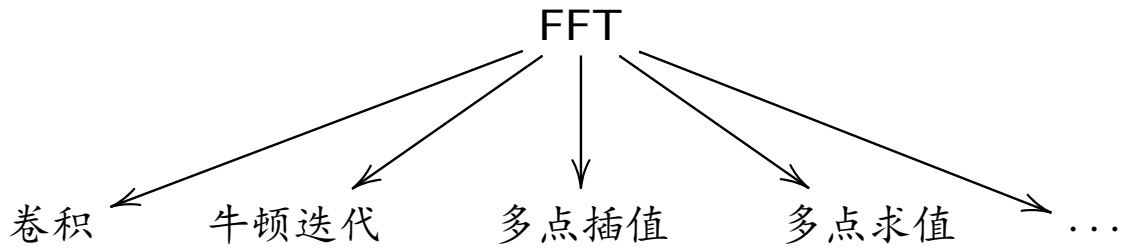
- 我的论文前两部分是FFT的基本介绍，FFT的基本应用，最后一部分介绍了一个不是很普及的技巧，其中引入了一个仅用4次实数DFT实现10^9级别的任意模数下的卷积的技巧。

Contents

- 我的论文前两部分是FFT的基本介绍，FFT的基本应用，最后一部分介绍了一个不是很普及的技巧，其中引入了一个仅用4次实数DFT实现10^9级别的任意模数下的卷积的技巧。
- 限于时间关系这里只介绍最后一部分。

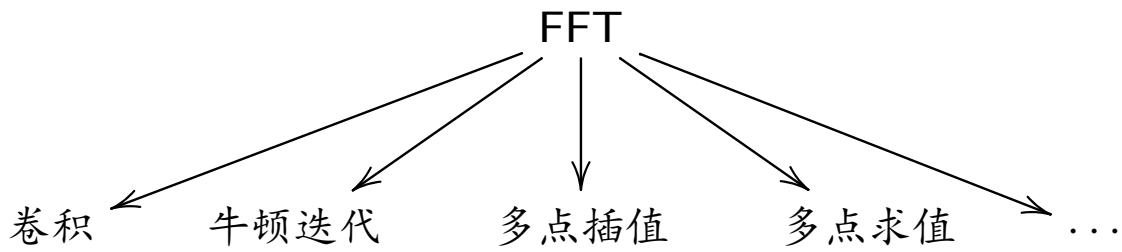
Introduction

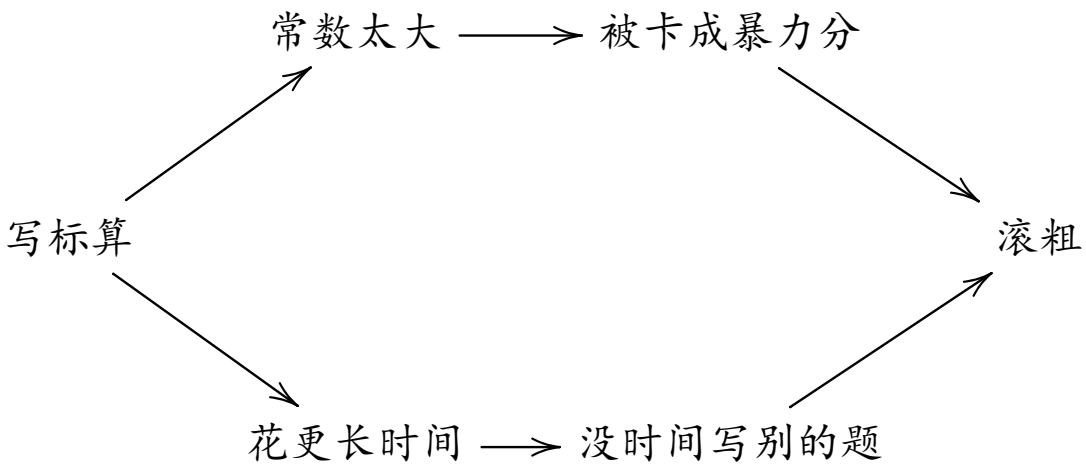
为什么要优化FFT的常数。

- 

Introduction

为什么要优化FFT的常数。

- 

- 

- 卷积的过程：序列长度增倍，对两个序列DFT，乘起来之后IDFT。

- 卷积的过程：序列长度增倍，对两个序列DFT，乘起来之后IDFT。
- 3次2倍长度的DFT。

- 卷积的过程：序列长度增倍，对两个序列DFT，乘起来之后IDFT。
- 3次2倍长度的DFT。
- 我们分两步做优化：2次2倍长度的DFT，3次1倍长度的DFT。

- 卷积的过程：序列长度增倍，对两个序列DFT，乘起来之后IDFT。
- 3次2倍长度的DFT。
- 我们分两步做优化：2次2倍长度的DFT，3次1倍长度的DFT。
- 直观来看，常数分别是原来的$\frac{2}{3}$，$\frac{1}{2}$。

- 接下来讨论DFT的次数时我们有时会不区分DFT和IDFT，比如3次IDFT加3次DFT被称为6次DFT。

- 接下来讨论DFT的次数时我们有时会不区分DFT和IDFT，比如3次IDFT加3次DFT被称为6次DFT。
- 我们考虑对长度为n的实多项式$A(x)$, $B(x)$进行DFT，假设n已经调整为2的整数次幂。

- 接下来讨论DFT的次数时我们有时会不区分DFT和IDFT，比如3次IDFT加3次DFT被称为6次DFT。
- 我们考虑对长度为n的实多项式$A(x)$, $B(x)$进行DFT，假设n已经调整为2的整数次幂。
- 我们定义：

$$\begin{aligned} P(x) &= A(x) + iB(x) \\ Q(x) &= A(x) - iB(x) \end{aligned}$$

- 接下来讨论DFT的次数时我们有时会不区分DFT和IDFT，比如3次IDFT加3次DFT被称为6次DFT。
- 我们考虑对长度为n的实多项式$A(x)$, $B(x)$进行DFT，假设n已经调整为2的整数次幂。
- 我们定义：

$$\begin{aligned} P(x) &= A(x) + iB(x) \\ Q(x) &= A(x) - iB(x) \end{aligned}$$

- 这里i指$\sqrt{-1}$，设$F_p[k], F_q[k]$分别表示对P和Q进行DFT之后得到序列的第k项，
即$F_p[k] = P(\omega^k), F_q[k] = Q(\omega^k)$，$\omega$是$n$次单位根。

- 接下来我们进行一系列推导：

- 接下来我们进行一系列推导：
- 由于排版问题我们用X代替$\frac{2\pi jk}{2L}$这个式子，每个X所对应的j,k的含义在上下文中可以看出。$\mathrm{conj}(x)$表示x的共轭复数。

- 接下来我们进行一系列推导：
- 由于排版问题我们用X代替$\frac{2\pi jk}{2L}$这个式子，每个X所对应的j,k的含义在上下文中可以看出。$\mathrm{conj}(x)$表示x的共轭复数。
-

$$\begin{aligned}
F_p[k] &= A(\omega_{2L}^k) + iB(\omega_{2L}^k) \\
&= \sum_{j=0}^{2L-1} A_j\omega_{2L}^{jk} + iB_j\omega_{2L}^{jk} \\
&= \sum_{j=0}^{2L-1} (A_j + iB_j)\left(\cos X + i\sin X\right) \\
&= \sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{kj}
\end{aligned}$$

$$
\begin{aligned}
F_q[k] &= A(\omega_{2L}^k) - iB(\omega_{2L}^k) \\
&= \sum_{j=0}^{2L-1} A_j \omega_{2L}^{jk} - iB_j \omega_{2L}^{jk} \\
&= \sum_{j=0}^{2L-1} (A_j - iB_j)(\cos X + i \sin X) \\
&= \sum_{j=0}^{2L-1} (A_j \cos X + B_j \sin X) + i (A_j \sin X - B_j \cos X) \\
&= \mathrm{conj} \left(\sum_{j=0}^{2L-1} (A_j \cos X + B_j \sin X) - i (A_j \sin X - B_j \cos X) \right)
\end{aligned}
$$

$$= \mathrm{conj}\left(\sum_{j=0}^{2L-1}\left(A_j\cos\left(-X\right)-B_j\sin\left(-X\right)\right)+i\left(A_j\sin\left(-X\right)+B_j\cos\left(-X\right)\right)\right)$$

$$= \mathrm{conj}\left(\sum_{j=0}^{2L-1}\left(A_j+iB_j\right)\left(\cos\left(-X\right)+i\sin\left(-X\right)\right)\right)$$

$$= \mathrm{conj}\left(\sum_{j=0}^{2L-1}\left(A_j+iB_j\right)\omega_{2L}^{-jk}\right)$$

$$= \mathrm{conj}\left(\sum_{j=0}^{2L-1}\left(A_j+iB_j\right)\omega_{2L}^{(2L-k)j}\right)$$

$$F_p[k] = \sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{kj}$$

$$F_q[k] = \text{conj}\left(\sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{(2L-k)j}\right)$$

- $$F_p[k] = \sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{kj}$$

$$F_q[k] = \text{conj}\left(\sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{(2L-k)j}\right)$$

- $F_q[k] = \text{conj}\left(F_p[2L-k]\right)$

- 于是我们仅用1次DFT就可以算出F_p和F_q。

- 于是我们仅用1次DFT就可以算出F_p和F_q。
- 令DFT($P[k]$)表示对$P(x)$进行DFT之后得到的序列的第k项，相信大家都知道DFT是线性变换，所以根据P, Q的定义：

- 于是我们仅用1次DFT就可以算出F_p和F_q。
- 令DFT($P[k]$)表示对$P(x)$进行DFT之后得到的序列的第k项，相信大家都知道DFT是线性变换，所以根据P, Q的定义：
-

$$\begin{aligned} F_p[k] &= \mathrm{DFT}(A[k]) + i\mathrm{DFT}(B[k]) \\ F_q[k] &= \mathrm{DFT}(A[k]) - i\mathrm{DFT}(B[k]) \end{aligned}$$

- 于是我们仅用1次DFT就可以算出F_p和F_q。
- 令DFT($P[k]$)表示对$P(x)$进行DFT之后得到的序列的第k项，相信大家都知道DFT是线性变换，所以根据P, Q的定义：
-

$$\begin{aligned} F_p[k] &= \mathrm{DFT}(A[k]) + i\mathrm{DFT}(B[k]) \\ F_q[k] &= \mathrm{DFT}(A[k]) - i\mathrm{DFT}(B[k]) \end{aligned}$$

- 接下来是激动人心的解二元一次方程：

$$\mathrm{DFT}(A[k]) = \frac{F_p[k] + F_q[k]}{2} \quad (1)$$

$$\mathrm{DFT}(B[k]) = i\frac{F_p[k] - F_q[k]}{2} \quad (2)$$

- 于是我们仅用1次DFT就可以算出F_p和F_q。
- 令DFT($P[k]$)表示对$P(x)$进行DFT之后得到的序列的第k项，相信大家都知道DFT是线性变换，所以根据P, Q的定义：
-

$$\begin{aligned} F_p[k] &= \mathrm{DFT}(A[k]) + i\mathrm{DFT}(B[k]) \\ F_q[k] &= \mathrm{DFT}(A[k]) - i\mathrm{DFT}(B[k]) \end{aligned}$$

- 接下来是激动人心的解二元一次方程：

$$\mathrm{DFT}(A[k]) = \frac{F_p[k] + F_q[k]}{2} \quad (1)$$

$$\mathrm{DFT}(B[k]) = i\frac{F_p[k] - F_q[k]}{2} \quad (2)$$

- 于是我们将2次DFT合并为了1次，可以减少1次DFT。

Optimization

- 我们能不能将2次DFT继续往下优化呢？当然能！

Optimization

- 我们能不能将2次DFT继续往下优化呢？当然能！
- 我们设$A_0(x)$是$A(x)$的偶次项的和，$A_1(x)$是奇次项的和，那么$A(x) = A_0(x^2) + xA_1(x^2)$。

假设我们要求$A(x)$与$B(x)$的乘积，且两个多项式的次数均为$n-1$（假设n为2的整数次幂），则有：

$$\begin{aligned} A(x)B(x) &= (A_0(x^2)+xA_0(x^2))(B_0(x^2)+xB_0(x^2)) \\ &= A_0(x^2)B_0(x^2)+xA_0(x^2)B_1(x^2)+xA_1(x^2)B_0(x^2)+x^2A_1(x^2)B_1(x^2) \\ &= A_0(x^2)B_0(x^2)+x(A_0(x^2)B_1(x^2)+A_1(x^2)B_0(x^2))+x^2A_1(x^2)B_1(x^2) \end{aligned}$$

$$A_0(x^2)B_0(x^2)+x(A_0(x^2)B_1(x^2)+A_1(x^2)B_0(x^2))+x^2A_1(x^2)B_1(x^2)$$

- 我们需要做4次多项式乘法，看作关于x^2的多项式，那么结果多项式的次数均不超过$n-1$。

$$A_0(x^2)B_0(x^2)+x(A_0(x^2)B_1(x^2)+A_1(x^2)B_0(x^2))+x^2A_1(x^2)B_1(x^2)$$

- 我们需要做4次多项式乘法，看作关于x^2的多项式，那么结果多项式的次数均不超过$n-1$。
- 还记得前面讲的合并DFT吗？只要2次DFT！

$$A_0(x^2)B_0(x^2)+x(A_0(x^2)B_1(x^2)+A_1(x^2)B_0(x^2))+x^2A_1(x^2)B_1(x^2)$$

- 我们需要做4次多项式乘法，看作关于x^2的多项式，那么结果多项式的次数均不超过$n-1$。
- 还记得前面讲的合并DFT吗？只要2次DFT！
- IDFT的话，式子右边的第一项和第三项，它们依然只有偶数项有系数。感觉就是能合并的！

$$A_0(x^2)B_0(x^2)+x(A_0(x^2)B_1(x^2)+A_1(x^2)B_0(x^2))+x^2A_1(x^2)B_1(x^2)$$

- 我们需要做4次多项式乘法，看作关于x^2的多项式，那么结果多项式的次数均不超过$n-1$。
- 还记得前面讲的合并DFT吗？只要2次DFT！
- IDFT的话，式子右边的第一项和第三项，它们依然只有偶数项有系数。感觉就是能合并的！
- 看成关于x^2的多项式，就是根据$A(x)$的DFT求$xA(x)$的DFT，$xA(x)$的DFT就是将$A(x)$的DFT的第k项乘上ω^k。

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？
- 并不一定所有项都是实数！ 

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？
- 并不一定所有项都是实数！ 
- 我们的老朋友：

$$\begin{aligned} F_p[k] &= \mathrm{IDFT}(A[k]) + i\mathrm{IDFT}(B[k]) \\ F_q[k] &= \mathrm{IDFT}(A[k]) - i\mathrm{IDFT}(B[k]) \end{aligned}$$

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？
- 并不一定所有项都是实数！ 
- 我们的老朋友：

$$\begin{aligned} F_p[k] &= \mathrm{IDFT}(A[k]) + i\mathrm{IDFT}(B[k]) \\ F_q[k] &= \mathrm{IDFT}(A[k]) - i\mathrm{IDFT}(B[k]) \end{aligned}$$

- IDFT之后的结果是实数！做完直接取实部和虚部！

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？
- 并不一定所有项都是实数！ 
- 我们的老朋友：

$$\begin{aligned} F_p[k] &= \mathrm{IDFT}(A[k]) + i\mathrm{IDFT}(B[k]) \\ F_q[k] &= \mathrm{IDFT}(A[k]) - i\mathrm{IDFT}(B[k]) \end{aligned}$$

- IDFT之后的结果是实数！做完直接取实部和虚部！
- 这样，IDFT的次数被减少到了1次。

- 哇！我们只要2次IDFT了啊，要是能合并IDFT多好？
- 并不一定所有项都是实数！ 
- 我们的老朋友：

$$\begin{aligned} F_p[k] &= \mathrm{IDFT}(A[k]) + i\mathrm{IDFT}(B[k]) \\ F_q[k] &= \mathrm{IDFT}(A[k]) - i\mathrm{IDFT}(B[k]) \end{aligned}$$

- IDFT之后的结果是实数！做完直接取实部和虚部！
- 这样，IDFT的次数被减少到了1次。
- 3次DFT，但是要做的序列长度只有原来的一半。

#34. 多项式乘法

描述 提交 自定义测试

这是一道模板题。

给你两个多项式，请输出乘起来后的多项式。

普通方法（多于500ms）：

ID	题目	提交者	结果	用时	内存	语言	文件大小	提交时间
#46959	#34. 多项式乘法	__debug	100	563ms	15684kb	C++	3.3kb	2016-02-03 16:35:50
#55790	#34. 多项式乘法	y0rkl1u	100	574ms	26868kb	C++	1.9kb	2016-03-21 08:15:57
#59884	#34. 多项式乘法	chongjg	100	577ms	14716kb	C++	2.5kb	2016-04-09 23:32:04
#5803	#34. 多项式乘法	fullpower	100	577ms	35332kb	C++	2.7kb	2015-01-25 20:22:28
#62685	#34. 多项式乘法	zhengtn03	100	578ms	21696kb	C++	3.9kb	2016-04-18 00:21:08
#53749	#34. 多项式乘法	NanoApe	100	580ms	9388kb	C++	1.6kb	2016-03-07 14:17:03
#41728	#34. 多项式乘法	xjt	100	582ms	11652kb	C++	1.2kb	2015-12-31 13:11:00
#45380	#34. 多项式乘法	Ciki	100	583ms	15108kb	C++	1.4kb	2016-01-23 20:06:07
#53622	#34. 多项式乘法	chiyich	100	584ms	19104kb	C++	2.0kb	2016-03-06 21:05:07
#38489	#34. 多项式乘法	lyxin65	100	589ms	10832kb	C++	2.0kb	2015-11-28 23:35:11

优化后的方法（最前面的几个是1.5次FFT，后面的几个是2次，300ms左右）：

ID	题目	提交者	结果	用时	内存	语言	文件大小	提交时间
#58866	#34. 多项式乘法	jcvb	100	255ms	21804kb	C++	12.4kb	2016-04-04 10:42:21
#58672	#34. 多项式乘法	poly_hacker	100	259ms	15772kb	C++	12.4kb	2016-04-02 21:22:03
#54294	#34. 多项式乘法	matthew99	100	279ms	11412kb	C++11	3.7kb	2016-03-11 10:54:17
#55085	#34. 多项式乘法	001	100	284ms	14876kb	C++	2.2kb	2016-03-17 16:50:36
#43431	#34. 多项式乘法	z52527	100	290ms	16300kb	C++	3.9kb	2016-01-10 22:15:02
#61996	#34. 多项式乘法	immortalCO	100	309ms	11188kb	C++	5.3kb	2016-04-16 15:18:21
#20504	#34. 多项式乘法	enot110	100	309ms	16292kb	C++11	3.9kb	2015-06-17 08:29:50
#54905	#34. 多项式乘法	alex_china	100	357ms	7272kb	C++11	3.1kb	2016-03-16 20:56:02
#54908	#34. 多项式乘法	Scape	100	358ms	7272kb	C++11	3.0kb	2016-03-16 20:57:28
#54985	#34. 多项式乘法	Saber	100	378ms	41112kb	C++	2.5kb	2016-03-16 22:04:07

Convolution taking modulo by arbitrary numbers

- 假设我们的模数是M，M是10^9级别。

Convolution taking modulo by arbitrary numbers

- 假设我们的模数是M，M是10^9级别。
- 我们设$M_0 = \lceil \sqrt{M} \rceil$，根据带余除法我们可以将所有整数$x$表示为$x = k[x]M_0 + b[x]$，其中$k[x]$和$b[x]$都是整数。

Convolution taking modulo by arbitrary numbers

- 假设我们的模数是M，M是10^9级别。
- 我们设$M_0 = \lceil \sqrt{M} \rceil$，根据带余除法我们可以将所有整数$x$表示为$x = k[x]M_0 + b[x]$，其中$k[x]$和$b[x]$都是整数。
- $597855228 \times 1294683923 =$
  $(18682 \times 32000 + 31228) \times (40458 \times 32000 + 27923)$

Convolution taking modulo by arbitrary numbers

- 假设我们的模数是M，M是10^9级别。
- 我们设$M_0 = \lceil \sqrt{M} \rceil$，根据带余除法我们可以将所有整数$x$表示为$x = k[x]M_0 + b[x]$，其中$k[x]$和$b[x]$都是整数。
- $597855228 \times 1294683923 =$
  $(18682 \times 32000 + 31228) \times (40458 \times 32000 + 27923)$
- 这样拆系数可以起到避免精度问题的作用。

- 我们假设多项式$A(x)$的系数序列为a_i，多项式$B(x)$的系数序列为b_i，那么我们把$k[a_i], b[a_i], k[b_i], b[b_i]$形成的四个序列两两做1次卷积。

- 我们假设多项式$A(x)$的系数序列为a_i，多项式$B(x)$的系数序列为b_i，那么我们把$k[a_i], b[a_i], k[b_i], b[b_i]$形成的四个序列两两做1次卷积。
-

$$\begin{aligned} & (A_0(x) \times 32000 + A_1(x))(B_0(x) \times 32000 + B_1(x)) \\ = \quad & 32000^2(A_0(x)B_0(x)) + 32000(A_0(x)B_1(x) + A_1(x)B_0(x)) + A_1(x)B_1(x) \end{aligned}$$

- 我们假设多项式$A(x)$的系数序列为a_i，多项式$B(x)$的系数序列为b_i，那么我们把$k[a_i], b[a_i], k[b_i], b[b_i]$形成的四个序列两两做1次卷积。
-

$$\begin{aligned} & (A_0(x) \times 32000 + A_1(x))(B_0(x) \times 32000 + B_1(x)) \\ = \quad & 32000^2(A_0(x)B_0(x)) + 32000(A_0(x)B_1(x) + A_1(x)B_0(x)) + A_1(x)B_1(x) \end{aligned}$$

- 预处理四个DFT，乘起来之后3个IDFT，用前面所述的合并DFT和IDFT的技巧，容易优化到4次。

一道FFT加快速幂的题目(Codeforces 623E Transforming Sequence):

- 3模数NTT: 3213ms

15854022	2016-02-07 11:33:52	matthew99	623E - Transforming Sequence	GNU C++11	Accepted	3213 ms	3000 KB

一道FFT加快速幂的题目(Codeforces 623E Transforming Sequence):

- 3模数NTT: 3213ms

15854022	2016-02-07 11:33:52	matthew99	623E - Transforming Sequence	GNU C++11	Accepted	3213 ms	3000 KB

- 无优化: 1996ms

16983551	2016-03-28 10:39:16	TakanashiRikka	623E - Transforming Sequence	GNU C++11	Accepted	1996 ms	6700 KB

一道FFT加快速幂的题目(Codeforces 623E Transforming Sequence):

- 3模数NTT: 3213ms

15854022	2016-02-07 11:33:52	matthew99	623E - Transforming Sequence	GNU C++11	Accepted	3213 ms	3000 KB

- 无优化: 1996ms

16983551	2016-03-28 10:39:16	TakanashiRikka	623E - Transforming Sequence	GNU C++11	Accepted	1996 ms	6700 KB

- 4次DFT: 420ms

15989812	2016-02-13 19:11:42	enot.1.10	623E - Transforming Sequence	GNU C++11	Accepted	420 ms	8100 KB

一道FFT加快速幂的题目(Codeforces 623E Transforming Sequence):

- 3模数NTT: 3213ms

15854022	2016-02-07 11:33:52	matthew99	623E - Transforming Sequence	GNU C++11	Accepted	3213 ms	3000 KB

- 无优化: 1996ms

16983551	2016-03-28 10:39:16	TakanashiRikka	623E - Transforming Sequence	GNU C++11	Accepted	1996 ms	6700 KB

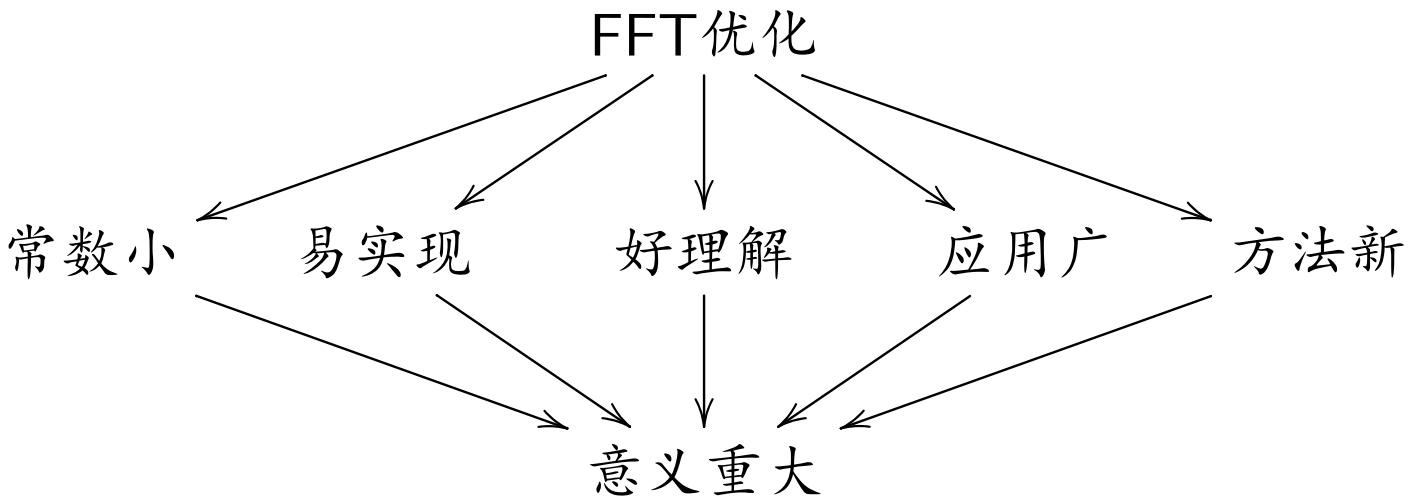
- 4次DFT: 420ms

15989812	2016-02-13 19:11:42	enot.1.10	623E - Transforming Sequence	GNU C++11	Accepted	420 ms	8100 KB

- 优化快速幂: 374ms

16044780	2016-02-16 06:38:50	matthew99	623E - Transforming Sequence	GNU C++11	Accepted	374 ms	16800 KB

Summary



Acknowledgements

感谢中国计算机学会提供学习和交流的平台。
感谢雅礼中学的汪星明老师多年来给予的关心和指导。
感谢大家的聆听。

Questions

欢迎提问。