Project Report

陈浩贤 18307110276

在这个 Project 中，主要做了以下工作：

- 实现了任意长度的 FFT 算法。针对长度 n 为合数和质数的情况，分别使用 Cooley-Tukey 和 Rader 算法进行实现。
- 实现了长度为 2 的幂次长度的 FFT 高效算法 Split Radix。

任意长度 FFT

Cooley-Tukey

Cooley-Tukey 是一种求解长度为合数的 FFT 算法。基 2 的 FFT 是一种特殊的 Cooley-Tukey，简单回顾一下：

$$
\begin{aligned}
&(E_0, E_1, \ldots, E_{n/2-1}) = \mathrm{FFT}(x_0, x_2, \ldots, x_{n-2}) \\
&(O_0, O_1, \ldots, O_{n/2-1}) = \mathrm{FFT}(x_1, x_3, \ldots, x_{n-1}) \\
&\text{for } k = 0, 1, \ldots, n/2-1: \ X_k = E_k + w_n^k O_k \\
&\text{for } k = n/2, n/2+1, \ldots, n-1: \ X_k = E_{k-n/2} + w_n^k O_{k-n/2}
\end{aligned}
$$

简单来说，就是将长度为 n 的 DFT 拆分成两个长度为 $n/2$ 的 DFT，从而使得计算的时间复杂度降低。

Cooley-Tukey 也是使用了这种分而治之的思想，根据长度的因数对序列进行拆分。假设 x 的长度为 N，其中 $N = N_1 N_2$，那么 DFT 的公式为

$$X_k = \sum_{n=0}^{N-1} \exp\left(-i\frac{2\pi}{N}nk\right) x_n$$

将以下两个式子代入上式

$$
\begin{aligned}
k = N_2 k_1 + k_2, \ k_1 \in \{0, 1, \ldots, N_1 - 1\}, \ k_2 \in \{0, 1, \ldots, N_2 - 1\} \\
n = N_1 n_2 + n_1, \ n_1 \in \{0, 1, \ldots, N_1 - 1\}, \ n_2 \in \{0, 1, \ldots, N_2 - 1\}
\end{aligned}
$$

可以得到

$$
\begin{aligned}
X_{N_2 k_1 + k_2} &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \exp\left(-i\frac{2\pi}{N_1 N_2}(N_1 n_2 + n_1)(N_2 k_1 + k_2)\right) x_{N_1 n_2 + n_1} \\
&= \sum_{n_1=0}^{N_1-1} \exp\left(-i\frac{2\pi}{N_1}n_1 k_1\right) \exp\left(-i\frac{2\pi}{N}n_1 k_2\right) \sum_{n_2=0}^{N_2-1} x_{N_1 n_2 + n_1} \exp\left(-i\frac{2\pi}{N_2}(n_2 k_2)\right)
\end{aligned}
$$

分析一下以上式子，$\sum\limits_{n_2=0}^{N_2-1} x_{N_1 n_2 + n_1} \exp\left(-i\frac{2\pi}{N_2}(n_2 k_2)\right)$ 正是长度为 N_2 的 DFT，对应的序列为 $x_{N_1 n_2 + n_1}, \ n_2 \in \{0, 1, \ldots, N_2 - 1\}$

记 $y_{n_1, k_2} = \sum\limits_{n_2=0}^{N_2-1} x_{N_1 n_2 + n_1} \exp\left(-i\frac{2\pi}{N_2}(n_2 k_2)\right), z_{n_1, k_2} = \exp\left(-i\frac{2\pi}{N}n_1 k_2\right) y_{n_1, k_2}$，则有

$$X_{N_2k_1+k_2}=\sum_{n_1=0}^{N_1-1}\exp\left(-i\frac{2\pi}{N_1}n_1k_1\right)z_{n_1,k_2}$$

这正是长度为 N_1 的 DFT。

根据以上的分析，对于 N 为合数的情况，我们可以先做 N_1 个长度为 N_2 的 DFT，然后每个元素乘上一个系数，再做 N_2 个长度为 N_1 的 DFT。那么时间复杂度可由下式表示：

$$T(N)=N_1T(N_2)+N_2T(N_1)+\Theta(N)$$

具体的算法步骤如下：

- 将长为 N 序列（向量）按列排列变为 $N_1\times N_2$ 的矩阵
- 对矩阵的每一行分别做 DFT
- 矩阵的每一个元素乘上系数 $\exp\left(-i\dfrac{2\pi}{N}n_1n_2\right)$，$n_1,n_2$ 分别表示该元素所处的行和列（从 0 开始）
- 对矩阵的每一列分别做 DFT
- 将 $N_1\times N_2$ 矩阵按行排列变为长为 N 的向量

推导思路如下：

记 $x[n_1][n_2]=x[N_1n_2+n_1]$，对每一行进行 DFT 变换，即固定 n_1，得到 $X_1[n_1][k_2]$，乘上系数后得到 $X_2[n_1][k_2]$，再对每一列进行 DFT 变换，固定k_2，得到 $X[k_1][k_2]=X[N_2k_1+k_2]$，所以最后按行排列即可得到原序列对应的 DFT。

Rader

对于长度为质数的情况，就不能将其简单的拆分了。注意到假如 N 为质数的话，那么 $N-1$ 一定为合数，如果能把一个元素单独抽出来，那么剩下就可以用 Cooley-Tukey 进行求解了。

Rader 算法需要用到数论的知识，首先介绍相关的知识。由于相关内容超出这门课的范围，就不详细展开证明，只进行叙述。

假设 N 为质数，定义 a,b 之间的运算 $\cdot$ 为 $(ab)\mod N$，则 $\{0,1,\ldots,N-1\}$ 和 $\cdot$ 可以构成一个群，这个群叫做整数模 N 乘法群。根据数论知识，这样的群存在一个整数 g，也叫做原根，使得对于任意非零的 n 都有唯一对应的 $q\in\{0,1,\ldots,N-2\}$，对应关系为 $n=g^p\mod N$。同样对于任意非零的 k 都有唯一对应的 $p\in\{0,1,\ldots,N-2\}$，使得 $k=g^{-p}\mod N$。此处，$k=g^{-p}\mod N$ 等价于 $kg^p\mod N=1$。

有了以上的结论，就可以将 $\{1,\ldots,N-1\}$ 映射到 $\{0,\ldots,N-2\}$。将上述 n，k 代入 DFT 的公式中，单独写出第 0 项，可以得到

$$X_0=\sum_{n=0}^{N-1}x_n$$

$$X_{g^{-p}(\mathrm{mod}N)}=x_0+\sum_{q=0}^{N-2}x_{g^q(\mathrm{mod}N)}\exp\left(-i\frac{2\pi}{N}g^{q-p}(\mathrm{mod}N)\right)$$

令 $a_q=x_{g^q(\mathrm{mod}N)}$，$b_q=\exp\left(-i\dfrac{2\pi}{N}g^{-q}(\mathrm{mod}N)\right)$，$q\in\{0,\ldots,N-2\}$，那么 $\sum_{q=0}^{N-2}x_{g^q(\mathrm{mod}N)}\exp\left(-i\dfrac{2\pi}{N}g^{q-p}(\mathrm{mod}N)\right)$ 实际上是 $\{a_q\}$ 和 $\{b_q\}$ 的循环卷积。根据循环卷积定理，有

$$a * b = (\hat{a} \cdot \hat{b})\check{}$$

这样循环卷积就可以转化为两个 DFT 和 一个 IDFT，这就转化成了我们熟悉的问题。

对于这种情况，为了方便进行 DFT，可以将其填充到 2 的幂次方，使用 基-2 的 FFT 算法进行求解。如果不将其补到 2 的幂次，使用 Rader 算法进行求解的过程中可能还会产生大的质数，这种情况下时间也会很大。

对于长度为 N 的两个序列，按以下方法进行补充：

- 首先找到一个最小的 $m \in \mathbb{Z}$，使得 $M = 2^m > 2N$
- 令 $\tilde{a} = [a_0, \underbrace{0, \dots, 0}_{M-N}, a_1, \dots, a_{N-1}]$, $\tilde{b} = [b_0, \dots, b_{N-1}, b_0, \dots]$ (一直重复 b，直到长度为 M，对超出的部分直接截断)
- 求 $\tilde{a} * \tilde{b}$，并取前 N 项，则得到 $a * b$

证明：

$$(\tilde{a} * \tilde{b})_k = \sum_{j=0}^{M-1} \tilde{a}_j \tilde{b}_{k-j} = a_0 b_k + \sum_{j=M-N}^{M-1} \tilde{a}_j \tilde{b}_{k-j} = a_0 b_k + \sum_{j=1}^{N-1} a_j \tilde{b}_{k+N-j} = \sum_{j=0}^{N-1} a_j b_{k-j} = (a * b)_k$$

长度为 2 的幂次长度的 FFT

DIT & DIF

在课上的，包括前面实现的 Cooley-Tukey 都是按时间抽取的 FFT，称为 Decimation-in-Time (DIT)。

下面介绍一种按频率进行抽取的基-2 FFT，称为 Decimation-in-Frequency (DIF)

$$\begin{aligned}
X_{2k} &= \sum_{n=0}^{N-1} \exp\left(-i\frac{2\pi}{N} 2kn\right) x_n \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N} 2kn\right) x_n + \sum_{n=N/2}^{N-1} \exp\left(-i\frac{2\pi}{N} 2kn\right) x_n \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N} 2kn\right) x_n + \exp\left(-i\frac{2\pi}{N} 2k(n+N/2)\right) x_{n+N/2} \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N} 2kn\right) (x_n + x_{n+N/2})
\end{aligned}$$

$$
\begin{aligned}
X_{2k+1} &= \sum_{n=0}^{N-1} \exp\left(-i\frac{2\pi}{N}(2k+1)n\right)x_n \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N}(2k+1)n\right)x_n + \sum_{n=N/2}^{N-1} \exp\left(-i\frac{2\pi}{N}(2k+1)n\right)x_n \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N}2kn\right)\exp\left(-i\frac{2\pi}{N}n\right)x_n + \exp\left(-i\frac{2\pi}{N}(2k+1)(n+N/2)\right)x_{n+N/2} \\
&= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N}2kn\right)\exp\left(-i\frac{2\pi}{N}n\right)(x_n - x_{n+N/2})
\end{aligned}
$$

于是可以将 $\{x_n + x_{n+N/2}\}_{n=0}^{N-1}, \{\exp\left(-i\frac{2\pi}{N}n\right)(x_n - x_{n+N/2})\}_{n=0}^{N-1}$ 看作两个新序列，对他们分别做 DFT。DIF 的步骤可以总结为

$$
\begin{aligned}
&(a_0, a_1, \ldots, a_{N/2-1}) = (x_0 + x_{N/2}, x_1 + x_{1+N/2}, \ldots, x_{N/2} + x_{N-1}) \\
&(b_0, b_1, \ldots, b_{N/2-1}) = (x_0 - x_{N/2}, x_1 - x_{1+N/2}, \ldots, x_{N/2} - x_{N-1}) \odot (w_N^0, w_N^1, \ldots, w_N^{N/2-1}) \\
&(X_0, X_2, \ldots, X_{N-2}) = \mathrm{FFT}(a_0, a_1, \ldots, a_{N/2-1}) \\
&(X_1, X_3, \ldots, X_{N-1}) = \mathrm{FFT}(b_0, b_1, \ldots, b_{N/2-1})
\end{aligned}
$$

DIF 与 DIT 区别在于 DIF 是先做加法和乘法，再做 FFT。而 DIT 正好相反。对于复序列来说计算量没有区别，但是对于实序列计算量存在差异，原因是 DIT 每次进行的 FFT 都是对实数进行操作的。

Split Radix

分裂基算法是基于 DIF 推导的，实际上是基-2和基-4的混合，对偶数下标的作基-2 FFT，对奇数下标的作基-4 FFT。具体如下：

$$
\begin{aligned}
X_{2k} &= \sum_{n=0}^{N/2-1} \exp\left(-i\frac{2\pi}{N}2kn\right)(x_n + x_{n+N/2}) \\
x_{4k+1} &= \sum_{n=0}^{N/4-1} \exp\left(-i\frac{2\pi}{N}4k\right)\exp\left(-i\frac{2\pi}{N}n\right)(x_n - x_{n+N/2} - i(x_{n+N/4} - x_{n+3N/4})) \\
x_{4k+3} &= \sum_{n=0}^{N/4-1} \exp\left(-i\frac{2\pi}{N}4k\right)\exp\left(-i\frac{2\pi}{N}3n\right)(x_n - x_{n+N/2} + i(x_{n+N/4} - x_{n+3N/4}))
\end{aligned}
$$

这样，长度为 N 的 DFT 可以分解为一个长度为 $N/2$ 和两个长度为 $N/4$ 的 DFT。分裂基算法相对于基-2的 FFT 计算量更小，原因是：可以观察到，偶数项前不需要乘旋转因子，这一部分计算量已经最小；而对于奇数项，由于计算机进行复数运算时是将实部和虚部分别进行计算，与 $\exp\left(i\pi k/2\right), k \in \mathbb{Z}$ 这类复数的乘积是可以简化的：$\exp\left(i\pi k\right), k \in \mathbb{Z}$ 不需要进行加法和乘法运算，只需要将符号或者实部虚部互换；$\exp\left(i\pi(k+\frac{1}{2})\right)$ 由于虚部和实部的绝对值相等，所以也只需进行两次加法运算和乘法运算；而其他复数则需要进行四次乘法和两次加法。

所以对于基-2 的 FFT，可以得到递推公式

$$
\mathrm{Add}(n) = \begin{cases} 4 & n = 2 \\ 16 & n = 4 \\ 2\mathrm{Add}(n/2) + 2n + 2(n/2-2) = 2\mathrm{Add}(n/2) + 3n - 4 & n \geq 8 \end{cases}
$$

$$
\mathrm{Mul}(n) = \begin{cases} 0 & n = 2, 4 \\ 2\mathrm{Mul}(n/2) + 4(n/2-4) + 2 \cdot 2 = 2\mathrm{Mul}(n/2) + 2n - 12 & n \geq 8 \end{cases}
$$

对于分裂基 FFT，可以得到递推公式

$$\mathrm{Add}(n)=\begin{cases}4 & n=2\\ 16 & n=4\\ \mathrm{Add}(n/2)+2\mathrm{Add}(n/4)+3n+2(n/2-2) & n\ge 8\end{cases}$$

$$\mathrm{Mul}(n)=\begin{cases}0 & n=2,4\\ \mathrm{Mul}(n/2)+2\mathrm{Mul}(n/4)+4(n/2-4)+4 & n\ge 8\end{cases}$$

通过递推公式可以证明分裂基 FFT 的计算量，不管是加法还是减法，都更少。原因是它更好地利用了较小的 n，同时也更好地利用了单位根的性质。

证明：

乘法是显然的，因为后面加的一项是一样的，而由计算复杂度为 $\Theta(n\log n)$，所以 $\mathrm{Mul}(n)\ge 2\mathrm{Mul}(n/2)$，前一项也更小

令 Add_1 表示基-2 FFT，Add_2 表示分裂基 FFT，运用数学归纳法，对于 $n=2,4$ 成立，假设对于 $2,4,\dots,n/2$ 都成立，则有

$$\mathrm{Add}_1(n)-\mathrm{Add}_2(n)\ge \mathrm{Add}_1(n/2)-2\mathrm{Add}_1(n/4)-n=\frac{3}{2}n-4-n=\frac{1}{2}n-4\ge 0$$

得证

以下展示一些 n 的运算量

加法运算数：

n	Radix-2	Split Radix
8	52	52
16	148	144
32	388	372
64	964	912

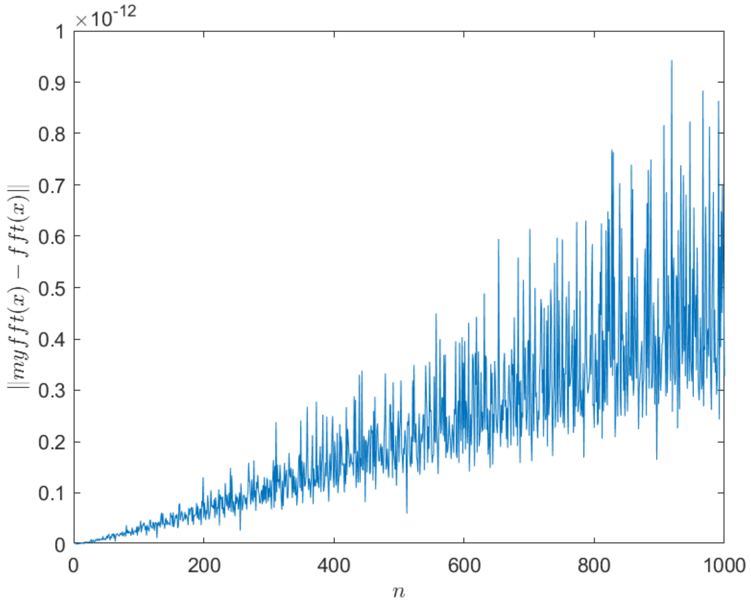
乘法运算数：

n	Radix-2	Split Radix
8	4	4
16	28	24
32	108	84
64	332	248

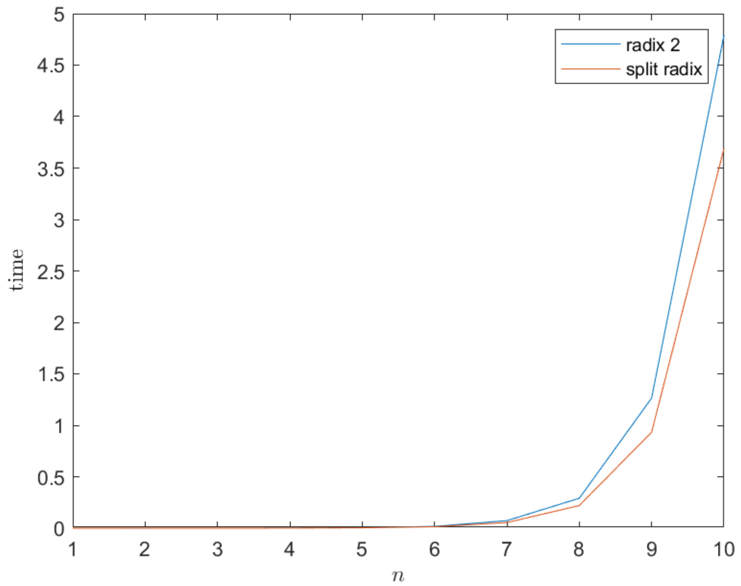
实验

算法准确性

与 MatLab 自带的 FFT 进行比较，结果如下：



基-2 与混合基算法时间比较



Reference

[1] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of computation, 1965, 19(90): 297-301.

[2] Rader C M. Discrete Fourier transforms when the number of data samples is prime[J]. Proceedings of the IEEE, 1968, 56(6): 1107-1108.

[3] Duhamel P, Hollmann H. Split radix'FFT algorithm[J]. Electronics letters, 1984, 20(1): 14-16.