

《算法设计与分析》

4-快速傅立叶变换 (Fast Fourier Transform)

杨启哲

上海师范大学信机学院计算机系

2023 年 9 月 30 日

- › 多项式乘法
- › 快速傅里叶变换的细节

多项式乘法

考察两个多项式：

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

$$B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_nx^n$$

其乘积 $C(x) = A(x) \cdot B(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{2n}x^{2n}$ ，并且满足：

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0 = \sum_{i=0}^k a_i b_{k-i}$$

因此利用上面的公式计算 c_k 需要 $O(k)$ 次操作，求出整个系数需要 $O(n^2)$ 的时间。

有没有更快的算法？

整数相乘的算法实际上严重依赖于多项式乘法的算法。

例如，我们想要计算 1234×5678 ，我们可以将其转化为多项式乘法的形式：

$$\begin{aligned}1234 \times 5678 &= (1 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 4) \times (5 \times 10^3 + 6 \times 10^2 + 7 \times 10 + 8) \\ &= A(10) \times B(10)\end{aligned}$$

其中 $A(x) = x^3 + 2x^2 + 3x + 4$, $B(x) = 5x^3 + 6x^2 + 7x + 8$.

从而计算两数相乘变成了计算多项式 $C(x) = A(x) \cdot B(x)$ ，再将 x 换成使用的基数即可。

n 次多项式可以通过 $n + 1$ 个系数来进行表示：

- $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ 可以简单表示为 $[a_0, \dots, a_n]$.
- 一个 $n + 1$ 维向量 $[a_0, \dots, a_n]$ 可以唯一确定一个多项式。

我们介绍其另一种表示方法，其运用到下列的性质：

事实 1.

一个 n 次多项式可以被任意其 $n + 1$ 个不同点处的取值所唯一确定。

上述事实提供了另一种多项式的表示方法：**插值表示法**。

多项式的另一种表示方法：插值表示法 (II)

令 $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$, x_0, \dots, x_n 是任意 $n + 1$ 个不同的实数, 则:

- $[A(x_0), \dots, A(x_n)]$ 可以唯一确定多项式 $A(x)$.

例 2.

考察 2 次多项式 $f(x)$, 其满足 $f(0) = 1, f(1) = 2, f(2) = 4$, 则:

$$\begin{aligned} f(x) &= 1 \cdot \frac{(x-1)(x-2)}{(0-1)(0-2)} + 2 \cdot \frac{(x-0)(x-2)}{(1-0)(1-2)} + 4 \cdot \frac{(x-0)(x-1)}{(2-0)(2-1)} \\ &= 1 \cdot \frac{x^2 - 3x + 2}{2} + 2 \cdot (-x^2 + 2x) + 4 \cdot \frac{x^2 - x}{2} \\ &= \frac{1}{2}x^2 + \frac{1}{2}x + 1 \end{aligned}$$

多项式的另一种表示方法：插值表示法 (III)

事实上，如果我们知道了一个至多为 n 次多项式在 $n + 1$ 个不同点处的取值 $(x_0, y_0), \dots, (x_n, y_n)$ ，则该多项式可以被下列方式唯一的计算出来：

$$f(x) = \sum_{i=0}^n y_i \cdot \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}$$

上述方法即拉格朗日插值法。

但我们不用管多项式的具体形式，目前我们已经有了两种来表示 n 次多项式的方法：

- 系数表示法：用其 $n + 1$ 个系数进行表示： $[a_0, \dots, a_n]$.
- 插值表示法：用其 $n + 1$ 个点来进行表示： $[A(x_0), \dots, A(x_n)]$.

对于两种表示方法来说,

- 系数表示法能够很直观的显示多项式的情形。
 - $A = [a_0, \dots, a_n] \implies A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n.$
- 插值表示法则能够更加方便的对多项式乘法进行计算。
 - 假设现在有 n 次多项式 $A(x)$ 和 $B(x)$, 我们要计算 $C(x) = A(x) \cdot B(x)$.
 - $C(x)$ 至多是 $2n$ 次多项式, 因此我们知道 $2n + 1$ 个点的取值便可以确定该多项式。
 - 通过 $A(x_0), \dots, A(x_{2n})$ 和 $B(x_0), \dots, B(x_{2n})$ 我们可以快速确定 $C(x)$ 中的 $2n + 1$ 个点, 这是因为:

$$C(x_i) = A(x_i) \cdot B(x_i), i = 0, \dots, 2n$$

由上述讨论可知，计算由值表示的多项式的乘法所需的时间是线性的（只需计算 $2n + 1$ 个乘积），从而我们不难想到如下的思路：

- 计算步骤：通过输入多项式的系数得到一组多项式在选定点处的值，并求得相应乘积在定点处的值。
- 插值步骤：通过多项式在选定点处的值还原出多项式的系数。

整个思路如下图所示：





算法: PolynomialMultiplication(A, B)

输入: 两个 n 次多项式的系数 $A = [a_0, \dots, a_n]$, $B = [b_0, \dots, b_n]$

输出: 两个多项式的乘积的系数表示 $C = [c_0, \dots, c_{2n}]$

选择阶段:

- 1: 选择 $2n + 1$ 个不同的点 x_0, \dots, x_{2n} .

计算阶段:

- 2: 计算 $A(x_0), \dots, A(x_{2n})$ 和 $B(x_0), \dots, B(x_{2n})$.

乘法阶段:

- 3: 通过 $C(z) = A(z) \cdot B(z)$ 计算 $C(x_0), \dots, C(x_{2n})$.

插值阶段:

- 4: 根据 $C(x_0), \dots, C(x_{2n})$ 还原出 C 的系数 $[c_0, \dots, c_{2n}]$.

正确性

该算法的正确性可以由多项式的两种表示方法的等价性得到。

复杂性

- 选择阶段显然只需要 $O(n)$ 的时间。
- 乘法阶段也只需要 $O(n)$ 的时间。
- 在计算阶段，由 Horner 规则可知，计算一个点的值需要 $O(n)$ 的时间，因此如果我们采用朴素的方法，计算阶段需要 $O(n^2)$ 的时间。
- 插值阶段我们暂时还没有讨论。

下面，我们先着重处理如何提升计算阶段的效率。

一个很重要的想法在于对于一个多项式 $A(x)$ 当我们计算 $A(x_0)$ 和 $A(-x_0)$ 时, 由于其偶次幂相同, 其实存在大量重复的运算。

考察如下的多项式:

$$A(x) = 3 + 4x + 6x^2 + 2x^3 + x^4 + 10x^5$$

我们将其划分成 x 的奇次幂项和偶次幂项, 即:

$$A(x) = (3 + 6x^2 + x^4) + x(4 + 2x^2 + 10x^4)$$

$A_1(x^2) = 3 + 6x^2 + x^4$ $A_2(x^2) = 4 + 2x^2 + 10x^4$

计算 $A(x)$ 在 $\pm 1, \pm 2, \pm 3$ 的值可以变为计算 $A_1(x), A_2(x)$ 在 $1, 4, 9$ 上的值!

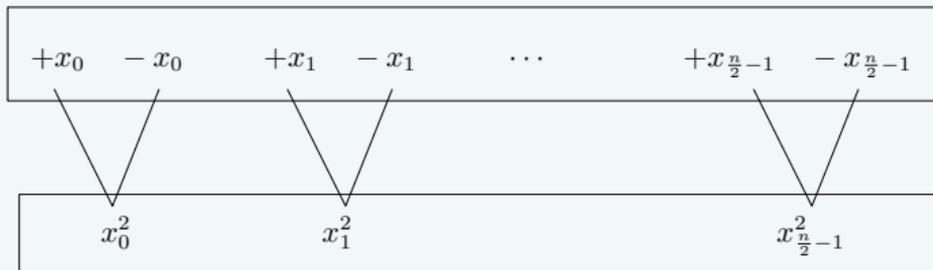
一般的, 对于一个 n 次多项式 $A(x)$ (为了方便讨论, 假设 n 是偶数), 我们可以将其转换为:

$$A(x) = A_e(x^2) + xA_o(x^2)$$

其中, $A_e(x)$, $A_o(x)$ 都是连个次数小于等于 $\frac{n}{2} - 1$ 的多项式。

则对于给定点对 $\pm x_0, \dots, \pm x_{\frac{n}{2}-1}$ 在 $A(x)$ 的计算, 其可以转换为对于点对 $x_0^2, \dots, x_{\frac{n}{2}-1}^2$ 在 $A_e(x)$, $A_o(x)$ 的计算。

计算 $A(x)$ 在:



等价于计算 $A_e(x)$, $A_o(x)$ 在:

按照上述的方法，规模为 n 的原问题被转移成为了 2 个规模为 $\frac{n}{2}$ ，从而上述思路得到的算法运行时间满足：

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

计算可得, $T(n) = O(n \log n)$ ，似乎我们已经完成了目标！

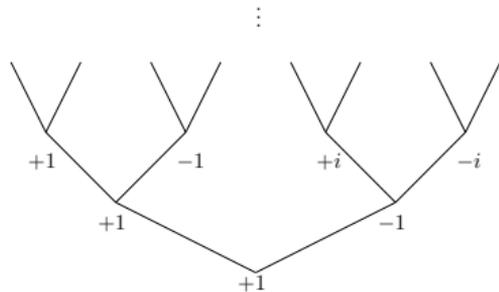
平方为负数？

上述方法似乎还面临一个问题，在计算的下一层，我们还需要 $x_0^2, \dots, x_{\frac{n}{2}-1}^2$ 也是成对出现的，可是一个平方数如何是负的？

$\sqrt{-1} = i$ ！我们需要使用**复数**！

我们对递归作反向工程来发现需要选取的复数:

- 在递归的最底层, 只会有一点 1.
- 在递归的下一层, 我们需要选取两个点 $\pm\sqrt{1} = \pm 1$.
- 而在下一层, 则是 $\pm\sqrt{1} = \pm 1$, $\pm\sqrt{-1} = \pm i$.
- ...



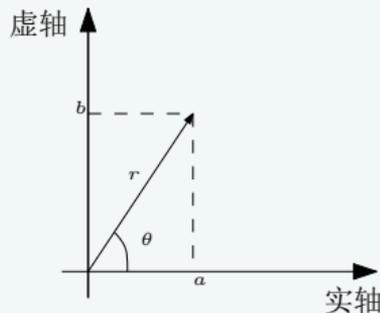
单位元的 n 次复根

随着递归的进行, 我们最终会获得的 n 个点会满足:

$$z^n = 1$$

复数与复平面

- 复数 $z = a + bi$ 可以视作复平面上的一个点。
- 极坐标表示: $z = r(\cos \theta + i \sin \theta) = re^{i\theta}$.
- 长度: $r = \sqrt{a^2 + b^2}$.
- 角度: $\theta \in [0, 2\pi]$ 满足: $\sin \theta = \frac{b}{r}$, $\cos \theta = \frac{a}{r}$.

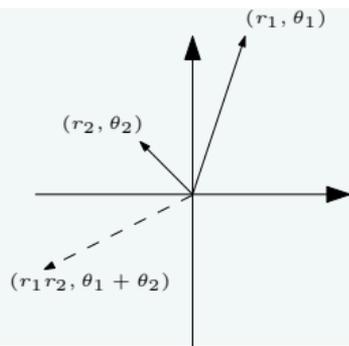


极坐标乘法的简便性

两个复数相乘等价于其长度相乘, 角度相加:

$$(r_1, \theta_1) \times (r_2, \theta_2) = (r_1 r_2, \theta_1 + \theta_2).$$

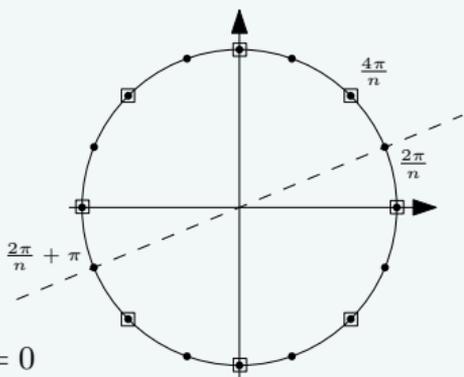
若 $z = (1, \theta)$ 即 z 在单位圆上, 则 $z^n = (1, n\theta)$.



单位元的 n 次复根

右图显示了方程 $z^n = 1$ 的解 ($n = 16$ 的情况):

- 上述方程的解是 $z = (1, \theta)$, 其中 θ 是 $\frac{2\pi}{n}$ 的倍数。
- 当 n 是偶数的时候:
 - 解正负成对出现, 带方框的点表示方程 $z^{\frac{n}{2}} = 1$ 的解。
- 令 $\omega = e^{i\frac{2\pi}{n}}$, 则我们有:
 - 若 i 不是 n 的倍数, 则 $1 + \omega^i + \omega^{2i} + \dots + \omega^{(n-1)i} = 0$
 - 若 i 是 n 的倍数, 则 $1 + \omega^i + \omega^{2i} + \dots + \omega^{(n-1)i} = n$ 。



计算阶段的分治算法

该想法的核心在于要计算 n 个 n 次单位根在 $A(x)$ 上的值, 等价于计算 $A_e(x)$ 和 $A_o(x)$ 在 $\frac{n}{2}$ 个 $\frac{n}{2}$ 次单位根上的值。

由上述讨论，我们终于得到了计算阶段的分治算法，也即快速傅里叶变换 (FFT):

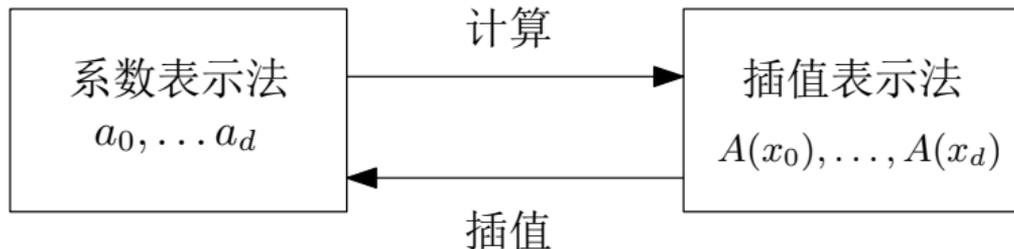
算法 FFT(A, ω)

输入: 系数表示的至多 $n - 1$ 次多项式 $A(x)$ ，这里假设 $n = 2^k$ ； ω 是一个 n 次单位根。

输出: $A(\omega^0), \dots, A(\omega^{n-1})$ 的值

- 1: **if** $n = 1$ **then**
- 2: **return** $A(\omega^0)$
- 3: 将 $A(x)$ 表示为 $A_e(x^2) + xA_o(x^2)$
- 4: $[A_e(\omega^2), \dots, A_e(\omega^{2n-2})] \leftarrow \text{FFT}(A_e, \omega^2)$
- 5: $[A_o(\omega^2), \dots, A_o(\omega^{2n-2})] \leftarrow \text{FFT}(A_o, \omega^2)$
- 6: **for** $j = 0, \dots, n - 1$ **do**
- 7: $A(\omega^j) \leftarrow A_e(\omega^{2j}) + \omega^j A_o(\omega^{2j})$
- 8: **return** $[A(\omega^0), \dots, A(\omega^{n-1})]$

回顾一下我们的思路:



现在只剩插值阶段了! 只需要通过计算好的 $C(\omega^0), \dots, C(\omega^{n-1})$ 还原出 C 的多项式即可!

神奇的结论!

算法 $\text{FFT}([C(\omega^0), \dots, C(\omega^{n-1})], \omega^{-1})$ 恰好计算出了所有的系数! 换句话说, 两个过程几乎是一样的!

我们下面来解释这个神奇的结论。首先我们回顾一下，我们之前究竟算了些什么。

通过算法的计算阶段和乘法阶段，我们算出了 $C(x_0), \dots, C(x_{N-1})$ 的具体值。这里为了后面的表示方便，我们不妨假定 $N-1$ 就是 C 的次数，因为显然后面的步骤与 $A(x)$ 跟 $B(x)$ 无关了。

另一方面，假设 $C(x) = c_0 + c_1x + \dots + c_{n-1}x^{n-1}$ ，则上述过程可以变成如下矩阵的计算形式：

$$\begin{bmatrix} C(x_0) \\ C(x_1) \\ \vdots \\ C(x_{N-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & \cdots & x_0^{N-1} \\ 1 & x_1 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & \cdots & x_{N-1}^{N-1} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix}$$

$$\begin{bmatrix} C(x_0) \\ C(x_1) \\ \vdots \\ C(x_{N-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & \cdots & x_0^{N-1} \\ 1 & x_1 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & \cdots & x_{N-1}^{N-1} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix}$$

我们令 M 表示上式中间的矩阵，这是一个著名的矩阵: Vandermonde 矩阵。

Vandermonde 矩阵

Vandermonde 矩阵具有非常奇妙的性质，下面是一些我们要用到的性质：

- 如果 x_0, \dots, x_{n-1} 互不相同，则 M 是可逆的。
- 计算 M 的逆矩阵需要 $O(n^2)$ 的时间。

所以计算过程是 M 。而插值过程其实就是 M 的逆 M^{-1} !

尽管 M 具有非常好的性质，但是我们还是需要 $O(n^2)$ 的时间来计算 M^{-1} ，这显然不是我们想要的。

让我们再将目光转回到复数上来，令 x_0, \dots, x_{N-1} 表示为 N 次单位根，即 $x_i = \omega^i$ ，则之前的 FFT 算法可以表示成计算了如下的列向量：

$$\begin{bmatrix} C(\omega^0) \\ C(\omega^1) \\ \vdots \\ C(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

我们记中间的矩阵为 $M_N(\omega)$ ，显然插值阶段的计算就是计算 $M_N(\omega)^{-1}$ 。

概念回顾

- **内积**: 在复数向量域 \mathbb{C}^n 上, 两个向量 $\mathbf{u} = (u_0, \dots, u_{n-1})$ 与 $\mathbf{v} = (v_0, \dots, v_{n-1})$ 的内积定义为:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=0}^{n-1} u_i \bar{v}_i$$

其中 \bar{v}_i 表示 v_i 的共轭复数, 即如果 $v_i = re^{i\theta}$, 则 $\bar{v}_i = re^{-i\theta}$ 。

- **正交**: 两个向量正交, 即其夹角为直角。当两个向量正交时, 其内积为 0。

矩阵 $M_N(\omega)$ 的重要观察结论

$M_N(\omega)$ 的列向量是彼此正交的。

我们称 $M_N(\omega)$ 中的 n 个列向量组成了一个新的**正交基**，这个基称为**Fourier 基**。

矩阵 $M_N(\omega)$ 变换的本质

对一个向量乘以 $M_N(\omega)$ 相当于将其从标准基下的坐标变换成为 Fourier 基下的坐标。

因此，求 $M_N(\omega)$ 的逆等价于将 Fourier 基下的坐标变换成为标准基下的坐标。

反演公式

$$M_N(\omega)^{-1} = \frac{1}{n} M_N(\omega^{-1}).$$

至此，我们解释清楚了这个神奇的结论，即还原插值的过程也等价于调用 FFT 算法。

算法: PolynomialMultiplication(A, B)

输入: 两个 n 次多项式的系数 $A = [a_0, \dots, a_n]$, $B = [b_0, \dots, b_n]$, 这里假设 $2n = 2^k - 1$.

输出: 两个多项式的乘积的系数表示 $C = [c_0, \dots, c_{2n}]$

1: $x \leftarrow \omega$.

▷ ω 是 $2n + 1$ 次单位根

2: $\text{val}A \leftarrow \text{FFT}(A, \omega)$.

3: $\text{val}B \leftarrow \text{FFT}(B, \omega)$.

▷ 计算阶段

4: $\text{val}C \leftarrow [0, \dots, 0]$

5: **for** $i = 0, \dots, 2n$ **do**

6: $\text{val}C[i] = \text{val}A[i] \cdot \text{val}B[i]$

▷ 乘法阶段

7: $C \leftarrow \frac{1}{n} \cdot \text{FFT}(\text{val}C, \omega^{-1})$.

▷ 插值阶段

8: **return** C



多项式乘法

- 两种表示方法。
 - 系数表示法。
 - 插值表示法。
- 乘法的方式。
 - 用值取计算乘法，用系数取表示多项式。
- 选值的计算。
 - 快速傅立叶变换 FFT。

快速傅里叶变换的细节

让我们再次强调 FFT 其实解决了如下的运算：

$$\begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

让我们通过对 $M_n(\omega)$ 的分析再次感受一下分治的思想。其实只需要将其分成奇数列和偶数列两个部分即可。请注意为了方便表示，计数是从 0 开始的。

$n = 4$ 的例子

考察 $M_4(\omega)$, 这里 ω 我们认作是 4 次单位根, 即 $\omega = e^{i\frac{\pi}{2}} = i$ 。对于 $M_4(\omega) \cdot C$ 的过程, 当我们将 $M_4(\omega)$ 的奇数列和偶数列分开来写可得:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega & \omega^3 \\ 1 & \omega^4 & \omega^2 & \omega^6 \\ 1 & \omega^6 & \omega^3 & \omega^9 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_2 \\ c_1 \\ c_3 \end{bmatrix}$$

我们来逐一计算一下每一列的值, 可以观察到这个计算跟 $M_2(\omega^2) = \begin{bmatrix} 1 & 1 \\ 1 & \omega^2 \end{bmatrix}$ 息息相关。

n = 4 的例子 (II)

- 第 0 行:

$$r_0 = 1 \cdot c_0 + 1 \cdot c_1 + 1 \cdot c_2 + 1 \cdot c_3 = M_2(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \end{bmatrix} (0) + \omega^0 \cdot M_2(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \end{bmatrix} (0).$$

- 第 2 行:

$$r_1 = 1 \cdot c_0 + 1 \cdot c_1 + 1 \cdot c_2 + 1 \cdot c_3 = M_2(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \end{bmatrix} (0) + \omega^1 \cdot M_2(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \end{bmatrix} (0)$$

- 第 1 行:

$$r_2 = 1 \cdot c_0 + 1 \cdot c_1 + 1 \cdot c_2 + 1 \cdot c_3 = M_2(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \end{bmatrix} (0) - \omega^0 \cdot M_2(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \end{bmatrix} (0)$$

- 第 3 行:

$$r_3 = 1 \cdot c_0 + 1 \cdot c_1 + 1 \cdot c_2 + 1 \cdot c_3 = M_2(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \end{bmatrix} (0) - \omega^1 \cdot M_2(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \end{bmatrix} (0).$$

由上述例子可以看到，对于 $M_n(\omega)$ （这里 ω 表示 n 次单位根 $e^{i\frac{2\pi}{n}}$ ）当我们将奇数列和偶数列分开来表示的时候，第 j 行 r_j 的计算实际上可以转化为下列表达式：

$$M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \\ \vdots \\ c_{n-2} \end{bmatrix} + \omega^j \cdot M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

注意到 $\omega^{j+\frac{n}{2}} = \omega^{\frac{n}{2}} \cdot \omega^j = -\omega^j$ 。因此对于 $M_n(\omega)$ 的计算，可以转化成两个规模为 $\frac{n}{2}$ 的子问题：

$$\bullet M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \\ \vdots \\ c_{n-2} \end{bmatrix} \text{ 和 } M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \\ \vdots \\ c_{n-1} \end{bmatrix}.$$

让我们简单总结一下上述的过程，如果令

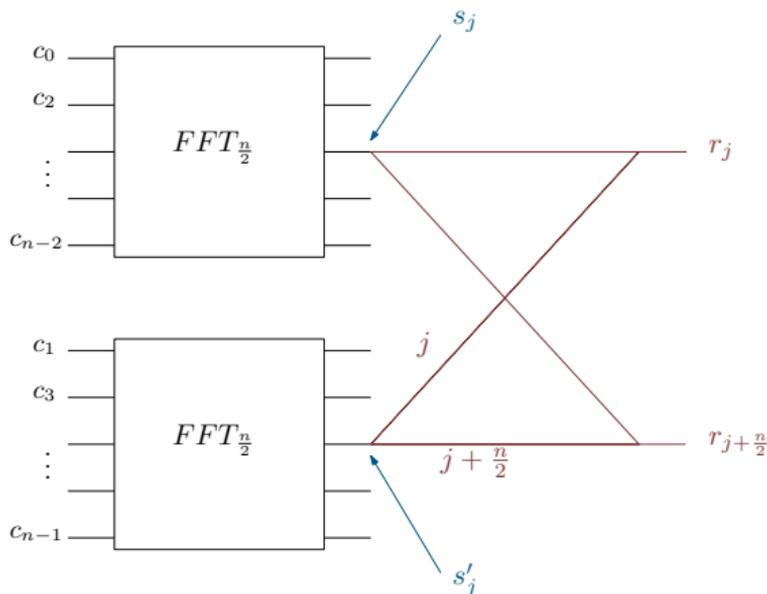
$$M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_0 \\ c_2 \\ \vdots \\ c_{n-2} \end{bmatrix} \cdot \text{算出的结果为} \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{\frac{n}{2}-1} \end{bmatrix}, M_{\frac{n}{2}}(\omega^2) \cdot \begin{bmatrix} c_1 \\ c_3 \\ \vdots \\ c_{n-1} \end{bmatrix} \cdot \text{算出的结果为} \begin{bmatrix} s'_0 \\ s'_1 \\ \vdots \\ s'_{\frac{n}{2}-1} \end{bmatrix}$$

则对于 $\forall j \in [0, 1, \dots, \frac{n}{2} - 1]$ ，我们有：

$$r_j = s_j + \omega^j s'_j \quad (1)$$

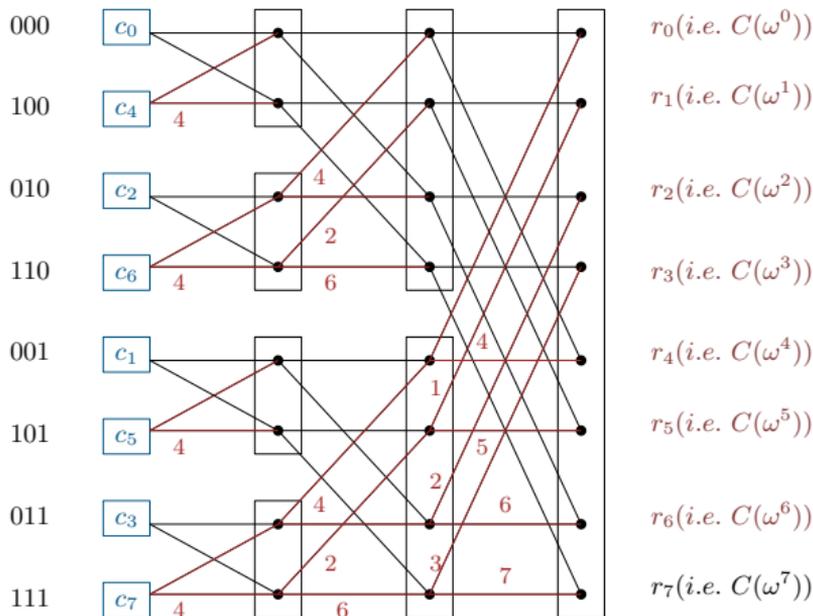
$$r_{j+\frac{n}{2}} = s_j - \omega^j s'_j \quad (2)$$

等式1和等式2可以用如下的蝴蝶式的线路样式表示：



- 边表示复数，两条边交汇表示将相应的复数进行相加。
- 边上的标记 j 表示对应的权重 ω^j 。

$n = 8$ 的展开



- 对于 n 个输入，一共有 $\log n$ 层，每层 n 个节点，共 $n \log n$ 次操作。
- 输入按特定形式排列，即按 $\log n$ 位的二进制从左往右计数排列。（ $n = 8$ 时为 0, 4, 2, 6, 1, 5, 3, 7）



本节内容

- 多项式的两种表示方法
 - 系数表示法。
 - 插值表示法。
- 多项式乘法算法-快速傅立叶变换
- 快速傅立叶变换的内部机制。