请问您今天要来点多项式吗？

11Dimensions

2020 年 7 月 29 日

多项式

一个环 R 上的关于 x 的多项式可以写作

$$A(x) = \sum_{i=0}^{n} a_i x^i$$

其中 $a_i \in R$。x 被称为这个多项式的自由元。

多项式

一个环 R 上的关于 x 的多项式可以写作

$$A(x) = \sum_{i=0}^{n} a_i x^i$$

其中 $a_i \in R$。x 被称为这个多项式的自由元。

多项式的次数被定义为其最高非零次项的次数，记为 $\deg A(x)$。

多项式运算

设 $A(x), B(x)$ 是次数不超过 n 的多项式。

多项式运算

设 $A(x), B(x)$ 是次数不超过 n 的多项式。

那么加法和减法运算被定义为：

$$A(x) \pm B(x) = \sum_{i=0}^{n} (a_i \pm b_i) x^i$$

多项式运算

设 $A(x), B(x)$ 是次数不超过 n 的多项式。

那么加法和减法运算被定义为：

$$A(x) \pm B(x) = \sum_{i=0}^{n} (a_i \pm b_i) x^i$$

显然可以在 $O(n)$ 时间内计算这两个多项式的和或差。

卷积

设 $\mathbf{a}, \mathbf{b}$ 是两个数列，那么这两个数列的卷积 $\mathbf{c}$ 的定义为

$$c_k = \sum_{i+j=k} a_i b_j$$

多项式乘法

两个多项式的乘积被定义为：

$$A(x)B(x) = \sum_{i=0}^{n}\sum_{j=0}^{n} a_i b_j x^{i+j} = \sum_{i=0}^{2n} c_k x^k$$

其中 $\mathbf{c}$ 是 $\mathbf{a}$ 与 $\mathbf{b}$ 的卷积。

多项式乘法

两个多项式的乘积被定义为：

$$A(x)B(x) = \sum_{i=0}^{n}\sum_{j=0}^{n} a_i b_j x^{i+j} = \sum_{i=0}^{2n} c_k x^k$$

其中 $\mathbf{c}$ 是 $\mathbf{a}$ 与 $\mathbf{b}$ 的卷积。朴素的按定义计算多项式乘法的时间复杂度是 $O(n^2)$ 的。

多项式的点值表示

给定一个不超过 n 次的多项式 $A(x)$ 以及 $n+1$ 个不同的点 $x_0, \ldots, x_n$，令 $y_i = A(x_i)$。

多项式的点值表示

给定一个不超过 n 次的多项式 $A(x)$ 以及 $n+1$ 个不同的点 $x_0, \ldots, x_n$，令 $y_i = A(x_i)$。

则这 $n+1$ 组 (x_i, y_i) 唯一的确定了这个多项式 $A(x)$。

多项式的点值表示

给定一个不超过 n 次的多项式 $A(x)$ 以及 $n+1$ 个不同的点 $x_0, \ldots, x_n$，令 $y_i = A(x_i)$。

则这 $n+1$ 组 (x_i, y_i) 唯一的确定了这个多项式 $A(x)$。

这些 (x_i, y_i) 称作这个多项式的点值表示。

如果给出 $A(x)$ 和 $B(x)$ 分别在 $x_0, \ldots, x_n$ 下的点值，则可以在 $O(n)$ 时间内得到 $A(x) \pm B(x)$ 或 $A(x)B(x)$ 在同一组位置处的点值。

系数与点值表示

给出 n 次多项式 $A(x)$ 的各项系数，可以通过求值计算多项式的点值表示。

给出 n 次多项式 $A(x)$ 的点值表示，可以通过待定系数，解方程得到多项式的各项系数。

系数与点值表示

给出 n 次多项式 $A(x)$ 的各项系数，可以通过求值计算多项式的点值表示。

给出 n 次多项式 $A(x)$ 的点值表示，可以通过待定系数，解方程得到多项式的各项系数。

如何在多项式的系数和点值表示之间快速转换？

系数与点值表示

给出 n 次多项式 $A(x)$ 的各项系数，可以通过求值计算多项式的点值表示。

给出 n 次多项式 $A(x)$ 的点值表示，可以通过待定系数，解方程得到多项式的各项系数。

如何在多项式的系数和点值表示之间快速转换？

这促使我们考虑一组特殊的点值。

单位根

满足 $x^n - 1 = 0$ 的 x 被称作 n 次单位根。

单位根

满足 $x^n - 1 = 0$ 的 x 被称作 n 次单位根。

设 ω 是 n 次单位根。如果 $\omega^0, \omega^1, \ldots, \omega^{n-1}$ 恰好生成了所有的 n 次单位根（即两两不同），则称 ω 为本原单位根。这等价于 n 是最小的使得 $\omega^n - 1 = 0$ 的正整数。我们用 ω_n 来表示一个 n 次本原单位根。

单位根

满足 $x^n - 1 = 0$ 的 x 被称作 n 次单位根。

设 ω 是 n 次单位根。如果 $\omega^0, \omega^1, \ldots, \omega^{n-1}$ 恰好生成了所有的 n 次单位根（即两两不同），则称 ω 为本原单位根。这等价于 n 是最小的使得 $\omega^n - 1 = 0$ 的正整数。我们用 ω_n 来表示一个 n 次本原单位根。

在复数域 $\mathbb{C}$ 上，$\omega_n = \exp\left(\frac{2\pi \mathrm{i}}{n}\right) = \cos\frac{2\pi}{n} + \mathrm{i}\sin\frac{2\pi}{n}$ 是一个本原单位根。下文首先考虑 $\mathbb{C}$ 上的多项式。

单位根

满足 $x^n - 1 = 0$ 的 x 被称作 n 次单位根。

设 ω 是 n 次单位根。如果 $\omega^0, \omega^1, \ldots, \omega^{n-1}$ 恰好生成了所有的 n 次单位根（即两两不同），则称 ω 为本原单位根。这等价于 n 是最小的使得 $\omega^n - 1 = 0$ 的正整数。我们用 ω_n 来表示一个 n 次本原单位根。

在复数域 $\mathbb{C}$ 上，$\omega_n = \exp\left(\frac{2\pi \mathrm{i}}{n}\right) = \cos\frac{2\pi}{n} + \mathrm{i}\sin\frac{2\pi}{n}$ 是一个本原单位根。下文首先考虑 $\mathbb{C}$ 上的多项式。

在有限域（即模素数的情况）中，本原单位根与数论中的原根有关。

离散傅里叶变换

设 $\mathbf{a}$ 是长度为 n 的数列，对 $0 \le k < n$，令

$$b_k = \sum_{i=0}^{n-1} a_i \cdot {\omega_n}^{ki}$$

则称 $\mathbf{b}$ 为 $\mathbf{a}$ 的离散傅里叶变换（DFT），记作 $\mathbf{b} = \mathcal{F}(\mathbf{a})$。

离散傅里叶变换

设 $\mathbf{a}$ 是长度为 n 的数列，对 $0 \leq k < n$，令

$$b_k = \sum_{i=0}^{n-1} a_i \cdot {\omega_n}^{ki}$$

则称 $\mathbf{b}$ 为 $\mathbf{a}$ 的离散傅里叶变换（DFT），记作 $\mathbf{b} = \mathcal{F}(\mathbf{a})$。

容易看出，令 $A(x) = \sum a_i x^i$，则 b_k 就是 $A(x)$ 在 ${\omega_n}^k$ 处的点值。因此计算 $\mathbf{a}$ 的 DFT 与计算 $A(x)$ 在 ${\omega_n}^0, {\omega_n}^1, \ldots, {\omega_n}^{n-1}$ 处的点值表示是等价的。

离散傅里叶变换的逆变换

对于长度为 n 的序列 $\mathbf{a}, \mathbf{b}$，回忆 DFT 的定义：

$$b_k = \sum_{i=0}^{n-1} a_i \cdot {\omega_n}^{ki} \quad (0 \le k < n) \tag{1}$$

离散傅里叶变换的逆变换

对于长度为 n 的序列 $\mathbf{a}, \mathbf{b}$，回忆 DFT 的定义：

$$b_k = \sum_{i=0}^{n-1} a_i \cdot {\omega_n}^{ki} \quad (0 \le k < n) \tag{1}$$

下面我们来证明 (1) 的逆变换（IDFT）如下：

$$a_k = \frac{1}{n} \sum_{i=0}^{n-1} b_i \cdot {\omega_n}^{-ki} \quad (0 \le k < n) \tag{2}$$

考虑将 (1) 代入 b_i，有

$$\begin{aligned}\sum_{i=0}^{n-1} b_i \cdot {\omega_n}^{-ki} &= \sum_{i=0}^{n-1} {\omega_n}^{-ki} \sum_{j=0}^{n-1} {\omega_n}^{ij} a_j \\ &= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} {\omega_n}^{-ki} \cdot {\omega_n}^{ij} \\ &= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} {\omega_n}^{i(j-k)}\end{aligned}$$

考虑将 (1) 代入 b_i，有

$$\begin{aligned}\sum_{i=0}^{n-1} b_i \cdot {\omega_n}^{-ki} &= \sum_{i=0}^{n-1} {\omega_n}^{-ki} \sum_{j=0}^{n-1} {\omega_n}^{ij} a_j \\ &= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} {\omega_n}^{-ki} \cdot {\omega_n}^{ij} \\ &= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} {\omega_n}^{i(j-k)}\end{aligned}$$

我们考虑式中的 $\sum_{i=0}^{n-1} {\omega_n}^{i(j-k)}$ 这一部分。

若 $j = k$，则

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

若 $j = k$，则

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

若 $j \neq k$，则由 $0 \leq j, k < n$，有 $|j - k| < n$，故 ${\omega_n}^{j-k} \neq 1$。因此由等比数列数列求和的结论得到

$$\begin{aligned}
\sum_{i=0}^{n-1} {\omega_n}^{i(j-k)} &= \sum_{i=0}^{n-1} \left({\omega_n}^{j-k}\right)^i \\
&= \frac{1 - \left({\omega_n}^{j-k}\right)^n}{1 - {\omega_n}^{j-k}} \\
&= \frac{1 - \left({\omega_n}^n\right)^{j-k}}{1 - {\omega_n}^{j-k}} = \frac{1-1}{1 - {\omega_n}^{j-k}} = 0
\end{aligned}$$

因此有

$$\begin{aligned}\frac{1}{n}\sum_{i=0}^{n-1} b_i \cdot {\omega_n}^{-ki} &= \frac{1}{n}\sum_{i=0}^{n-1} {\omega_n}^{-ki} \sum_{j=0}^{n-1} {\omega_n}^{ij} a_j \\ &= \frac{1}{n}\sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} {\omega_n}^{i(j-k)} \\ &= \frac{1}{n} \cdot n a_k = a_k\end{aligned}$$

即 (2) 成立。

类似的，可以证明由 (2) 成立可以推出 (1) 成立，故这两式是互逆的。这就是 IDFT。

类似的，可以证明由 (2) 成立可以推出 (1) 成立，故这两式是互逆的。这就是 IDFT。

IDFT 是 DFT 的逆变换，也就意味着已知多项式在单位根处的点值，IDFT 能够求出多项式的各项系数。在这种意义上，这个过程也可以看作插值。

循环卷积

对于两个长度为 n 的序列 $\mathbf{a}, \mathbf{b}$，定义

$$c_k = \sum_{(i+j) \bmod n = k} a_i b_j$$

则称 $\mathbf{c}$ 为 $\mathbf{a}$ 与 $\mathbf{b}$ 的循环卷积，记为 $\mathbf{c} = \mathbf{a} * \mathbf{b}$。

循环卷积

对于两个长度为 n 的序列 $\mathbf{a}, \mathbf{b}$，定义

$$c_k = \sum_{(i+j) \bmod n = k} a_i b_j$$

则称 $\mathbf{c}$ 为 $\mathbf{a}$ 与 $\mathbf{b}$ 的循环卷积，记为 $\mathbf{c} = \mathbf{a} * \mathbf{b}$。

关于循环卷积与 DFT，我们有如下定理：

$$\mathcal{F}(\mathbf{a} * \mathbf{b}) = \mathcal{F}(\mathbf{a}) \cdot \mathcal{F}(\mathbf{b})$$

其中 $\cdot$ 表示逐点乘积。

快速傅里叶变换

按定义，我们可以 $O(n^2)$ 实现 DFT 或 IDFT。快速傅里叶变换是快速计算 DFT 的方法，时间复杂度为 $O(n \log n)$。

快速傅里叶变换

按定义，我们可以 $O(n^2)$ 实现 DFT 或 IDFT。快速傅里叶变换是快速计算 DFT 的方法，时间复杂度为 $O(n \log n)$。

当 n 为 2 的幂次的时候，我们可以使用 Cooley–Tukey 算法来实现 FFT。

单位根的一些性质

考虑 ω_n 与 ω_{2n}，我们有：

$$\left({\omega_{2n}}^k\right)^2 = {\omega_n}^k$$

$${\omega_{2n}}^{n+k} = -{\omega_{2n}}^k$$

设 $n = 2m$。我们考虑将 $A(x)$ 的项按次数的奇偶性分类：

$$\begin{aligned} A(x) = \sum_{0 \le i < n} a_i x^i &= \sum_{0 \le i < m} a_{2i} x^{2i} + \sum_{0 \le i < m} a_{2i+1} x^{2i+1} \\ &= \sum_{0 \le i < m} a_{2i} x^{2i} + x \sum_{0 \le i < m} a_{2i+1} x^{2i} \end{aligned}$$

定义

$$A_0(x) = \sum_{0 \le i < m} a_{2i} x^i$$

$$A_1(x) = \sum_{0 \le i < m} a_{2i+1} x^i$$

定义

$$A_0(x) = \sum_{0 \le i < m} a_{2i} x^i$$

$$A_1(x) = \sum_{0 \le i < m} a_{2i+1} x^i$$

那么 $A_0(x), A_1(x)$ 都是次数不超过 m 的多项式，并且有

$$A(x) = A_0(x^2) + xA_1(x^2)$$

蝴蝶操作

结合单位根的性质，对于 $0 \leq k < m$，我们有

$$\begin{aligned} A({\omega_n}^k) &= A_0(({\omega_n}^k)^2) + {\omega_n}^k A_1(({\omega_n}^k)^2) \\ &= A_0({\omega_m}^k) + {\omega_n}^k A_1({\omega_m}^k) \\ A({\omega_n}^{m+k}) &= A_0(({\omega_n}^{m+k})^2) + {\omega_n}^{m+k} A_1(({\omega_n}^{m+k})^2) \\ &= A_0({\omega_m}^k) - {\omega_n}^k A_1({\omega_m}^k) \end{aligned}$$

蝴蝶操作

结合单位根的性质，对于 $0 \leq k < m$，我们有

$$\begin{aligned} A({\omega_n}^k) &= A_0(({\omega_n}^k)^2) + {\omega_n}^k A_1(({\omega_n}^k)^2) \\ &= A_0({\omega_m}^k) + {\omega_n}^k A_1({\omega_m}^k) \\ A({\omega_n}^{m+k}) &= A_0(({\omega_n}^{m+k})^2) + {\omega_n}^{m+k} A_1(({\omega_n}^{m+k})^2) \\ &= A_0({\omega_m}^k) - {\omega_n}^k A_1({\omega_m}^k) \end{aligned}$$

以上两式常被称为蝴蝶操作。

快速傅里叶变换

通过蝴蝶操作的过程可以看出，如果我们得到了 $A_0(x), A_1(x)$ 在点 ${\omega_m}^0, {\omega_m}^1, \ldots, {\omega_m}^{m-1}$ 处的点值，我们可以在 $O(n)$ 时间内计算出 $A(x)$ 在 ${\omega_n}^0, {\omega_n}^1, \ldots, {\omega_n}^{n-1}$ 处的点值。

快速傅里叶变换

通过蝴蝶操作的过程可以看出，如果我们得到了 $A_0(x), A_1(x)$ 在点 ${\omega_m}^0, {\omega_m}^1, \ldots, {\omega_m}^{m-1}$ 处的点值，我们可以在 $O(n)$ 时间内计算出 $A(x)$ 在 ${\omega_n}^0, {\omega_n}^1, \ldots, {\omega_n}^{n-1}$ 处的点值。

而计算 A_0, A_1 的点值的过程可以递归分治进行。

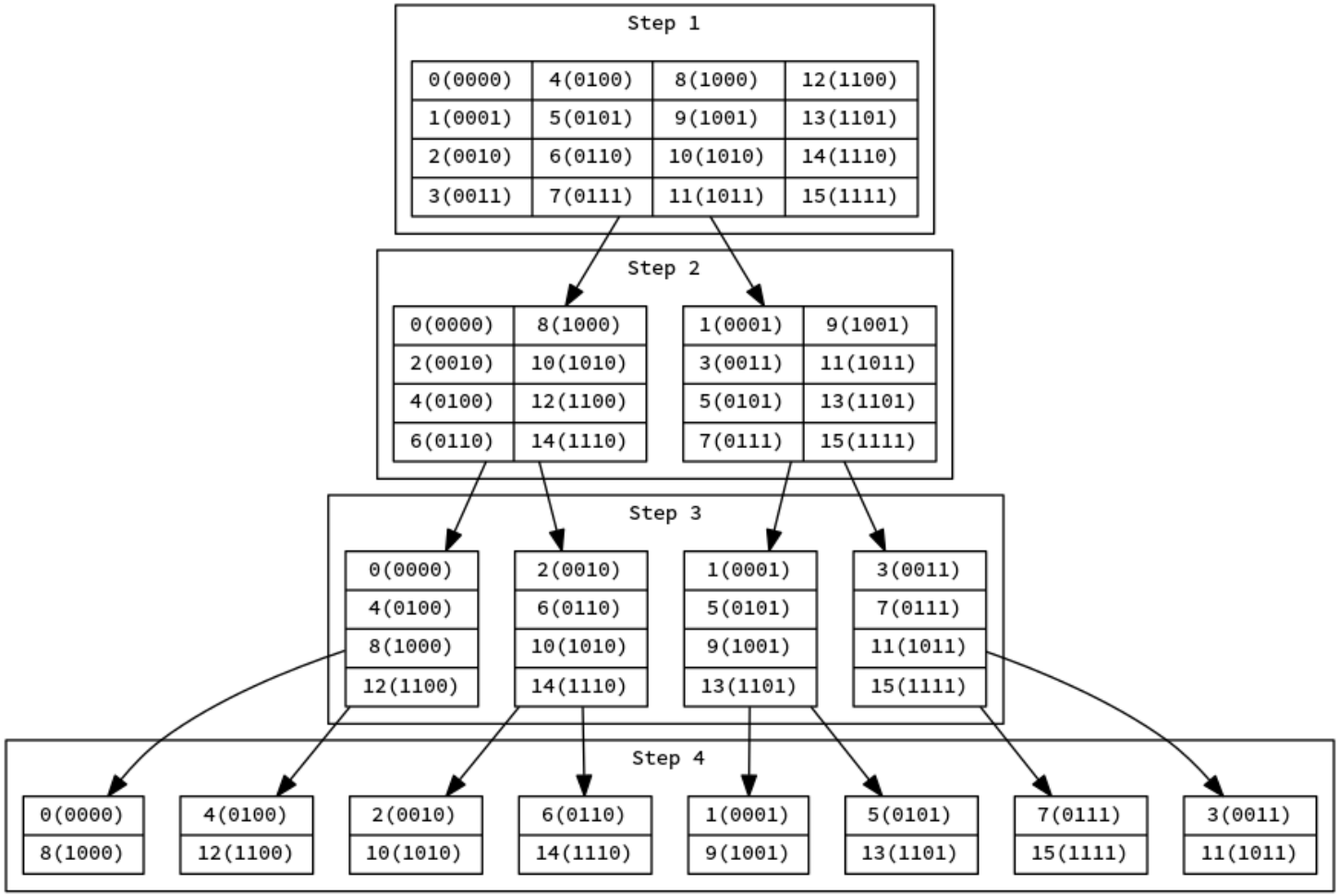
快速傅里叶变换

通过蝴蝶操作的过程可以看出，如果我们得到了 $A_0(x), A_1(x)$ 在点 ${\omega_m}^0, {\omega_m}^1, \ldots, {\omega_m}^{m-1}$ 处的点值，我们可以在 $O(n)$ 时间内计算出 $A(x)$ 在 ${\omega_n}^0, {\omega_n}^1, \ldots, {\omega_n}^{n-1}$ 处的点值。

而计算 A_0, A_1 的点值的过程可以递归分治进行。

根据主定理，我们可以在 $O(n \log n)$ 的时间内求出所要的 $A(x)$ 的点值表示。

为了快速的进行 IDFT，对比 DFT 与 IDFT 的表达式，可以发现我们只需要将 FFT 过程中用到的 ω_n 全部替换为 ${\omega_n}^{-1}$，最后再乘以 $\frac{1}{n}$ 即可。


Step 1
0(0000) 4(0100) 8(1000) 12(1100)
1(0001) 5(0101) 9(1001) 13(1101)
2(0010) 6(0110) 10(1010) 14(1110)
3(0011) 7(0111) 11(1011) 15(1111)
Step 2
0(0000) 8(1000) 1(0001) 9(1001)
2(0010) 10(1010) 3(0011) 11(1011)
4(0100) 12(1100) 5(0101) 13(1101)
6(0110) 14(1110) 7(0111) 15(1111)
Step 3
0(0000) 2(0010) 1(0001) 3(0011)
4(0100) 6(0110) 5(0101) 7(0111)
8(1000) 10(1010) 9(1001) 11(1011)
12(1100) 14(1110) 13(1101) 15(1111)
Step 4
0(0000) 4(0100) 2(0010) 6(0110) 1(0001) 5(0101) 7(0111) 3(0011)
8(1000) 12(1100) 10(1010) 14(1110) 9(1001) 13(1101) 15(1111) 11(1011)

位逆序置换

可以观察到，令 $\mathrm{rev}(x)$ 表示将 x 的二进制位的顺序反转之后得到的数字，令

$$a'_i = a_{\mathrm{rev}(i)}$$

则每次需要对 $\mathbf{a}$ 进行的蝴蝶操作在 $\mathbf{a}'$ 中变成了对两个相邻的序列的操作。

位逆序置换

可以观察到，令 $\mathrm{rev}(x)$ 表示将 x 的二进制位的顺序反转之后得到的数字，令

$$a'_i = a_{\mathrm{rev}(i)}$$

则每次需要对 $\mathbf{a}$ 进行的蝴蝶操作在 $\mathbf{a}'$ 中变成了对两个相邻的序列的操作。

把 $\mathbf{a}$ 转化为 $\mathbf{a}'$ 的过程常称为位逆序置换。

非递归的 FFT 实现

因此得到 $\mathbf{a}'$ 后，我们首先对 $\mathbf{a}'$ 的每一对相邻的长度为 1 的子序列做蝴蝶操作，然后对每一对相邻的长度为 2 的子序列……最后对两个长度为 $n/2$ 的子序列做蝴蝶操作，我们就完成了对 a 的 FFT。

DFT 与 FFT 都是在 $\mathbb{C}$ 中进行的过程。

DFT 与 FFT 都是在 $\mathbb{C}$ 中进行的过程。

在很多时候，我们往往是在对整数进行操作，并且经常要对某个素数 p 取模。

DFT 与 FFT 都是在 $\mathbb{C}$ 中进行的过程。

在很多时候，我们往往是在对整数进行操作，并且经常要对某个素数 p 取模。

注意到单位根在 DFT 中起了重要的作用，我们来考虑在模素数的时候是否存在和单位跟性质类似的元素。

原根

设 p 是素数。由费马小定理，我们知道对于任意 a 满足 $p \nmid a$，有

$$a^{p-1} \equiv 1 \pmod{p}$$

原根

设 p 是素数。由费马小定理，我们知道对于任意 a 满足 $p \nmid a$，有

$$a^{p-1} \equiv 1 \pmod{p}$$

g 称为模 p 的原根，当且仅当 $g^0, g^1, \ldots, g^{p-2}$ 在模 p 意义下互不相同。

原根

设 p 是素数。由费马小定理，我们知道对于任意 a 满足 $p \nmid a$，有

$$a^{p-1} \equiv 1 \pmod{p}$$

g 称为模 p 的原根，当且仅当 $g^0, g^1, \ldots, g^{p-2}$ 在模 p 意义下互不相同。

可以证明，模质数的原根总是存在的。

原根

设 p 是素数。由费马小定理，我们知道对于任意 a 满足 $p \nmid a$，有

$$a^{p-1} \equiv 1 \pmod{p}$$

g 称为模 p 的原根，当且仅当 $g^0, g^1, \ldots, g^{p-2}$ 在模 p 意义下互不相同。

可以证明，模质数的原根总是存在的。

原根的性质和本原单位根非常类似。换句话说，在 $\bmod p$ 意义下，g 可以被看做一个 $p-1$ 次本原单位根。

数论变换

设 n 满足 $n \mid p-1$。令 $\omega_n = g^{\frac{p-1}{n}}$。那么有

$$\omega_n{}^n = \left(g^{\frac{p-1}{n}}\right)^n = g^{p-1} \equiv 1 \pmod{p}$$

并且 $\omega_n{}^0, \omega_n{}^1, \ldots, \omega_n{}^{n-1}$ 互不相同。

数论变换

设 n 满足 $n \mid p-1$。令 $\omega_n = g^{\frac{p-1}{n}}$。那么有

$$\omega_n{}^n = \left(g^{\frac{p-1}{n}}\right)^n = g^{p-1} \equiv 1 \pmod{p}$$

并且 $\omega_n{}^0, \omega_n{}^1, \ldots, \omega_n{}^{n-1}$ 互不相同。

于是 ω_n 在 $\bmod p$ 意义下具有 n 次本原单位根的性质。我们可以利用它类似的定义 DFT。这被称为数论变换（NTT）。

快速数论变换

如果 $n = 2^k$，则也可以利用与 FFT 类似的方式快速的计算数论变换。

快速数论变换

如果 $n = 2^k$，则也可以利用与 FFT 类似的方式快速的计算数论变换。

但是因为 $2^k = n \mid p - 1$，这意味着快速数论变换对所选取的素数模数有着特殊的要求。

快速数论变换

如果 $n = 2^k$，则也可以利用与 FFT 类似的方式快速的计算数论变换。

但是因为 $2^k = n \mid p - 1$，这意味着快速数论变换对所选取的素数模数有着特殊的要求。

比如常见的模数

$$p_{\mathrm{UOJ}} = 998244353 = 7 \cdot 17 \cdot 2^{23} + 1$$

就是一个可以用于快速数论变换的模数。

FFT 的基本应用

FFT 计算的是 DFT，因此 FFT 的直接应用主要与卷积有关。

- 直接计算卷积
- 进行多项式相关运算
- 卷积与字符串匹配

利用 FFT 计算卷积

注意到利用 FFT，我们能直接进行的是长度为 2^k 的循环卷积。

利用 FFT 计算卷积

注意到利用 FFT，我们能直接进行的是长度为 2^k 的循环卷积。如果要进行一般的卷积运算，注意到两个长度为 n 的序列的卷积长度为 $2n-1$，因此一般会选择 k 使得 $2^k \geq 2n-1$，然后对序列进行长度为 2^k 的 FFT。

利用 FFT 计算卷积

注意到利用 FFT，我们能直接进行的是长度为 2^k 的循环卷积。

如果要进行一般的卷积运算，注意到两个长度为 n 的序列的卷积长度为 $2n-1$，因此一般会选择 k 使得 $2^k \geq 2n-1$，然后对序列进行长度为 2^k 的 FFT。

如果要进行一般的长度的循环卷积，则使用普通卷积来模拟。

利用 FFT 进行多项式乘法

多项式乘法就是系数进行卷积的过程，因此可以使用 FFT 计算。

利用 FFT 进行高精度乘法

高精度乘法就是多项式乘法，只需处理进位即可。

[ZJOI2014] 力

有一个长度为 n 的序列 (q_i)，定义

$$F_i = \sum_{j<i} \frac{q_i q_j}{(i-j)^2} - \sum_{j>i} \frac{q_i q_j}{(i-j)^2}$$

对于 $1 \le i \le n$，求 $E_i = F_i/q_i$。

$n \le 10^5, q_i \le 10^9$。

[ZJOI2014] 力

可以发现，答案可以用 $(\ldots,-\frac{1}{2^2},-\frac{1}{1^2},0,\frac{1}{1^2},\frac{1}{2^2},\frac{1}{3^2},\ldots)$ 与 (q_i) 的卷积表示。

使用 FFT 计算卷积即可。

[HNOI2017] 礼物

有两个长度为 n 的环形的正整数列 $\mathbf{x}, \mathbf{y}$，以及如下两个操作：

- 旋转（循环移位）其中一个数列
- 将其中一个数列的每个数增大一个相同的非负整数 c；

你可以进行每个操作至多一次，求你能使差异值 $\sum(x_i - y_i)^2$ 达到的最小值。

$n \leq 50000, x_i, y_i \leq 100$。

[HNOI2017] 礼物

假设已经进行完了旋转操作。考虑将某个数列增大一个非负整数后，差异值可以表示为

$$\sum (x_i - y_i + c)^2 = nc^2 + 2c\left(\sum x_i - \sum y_i\right) + \sum x_i^2 + \sum y_i^2 - 2\sum x_i y_i$$

其中 c 可正可负。

[HNOI2017] 礼物

假设已经进行完了旋转操作。考虑将某个数列增大一个非负整数后，差异值可以表示为

$$\sum (x_i - y_i + c)^2 = nc^2 + 2c\left(\sum x_i - \sum y_i\right) + \sum x_i^2 + \sum y_i^2 - 2\sum x_i y_i$$

其中 c 可正可负。

这是一个关于 c 的二次函数，最小值在 $c_0 = -\frac{1}{n}\left(\sum x_i - \sum y_i\right)$ 处取到，c 取一个最接近 c_0 的整数即可。

[HNOI2017] 礼物

容易发现旋转操作只会影响差异值表达中的最后一项。

[HNOI2017] 礼物

容易发现旋转操作只会影响差异值表达中的最后一项。

如果 $\mathbf{y}$ 旋转了 k 个位置，这一项用初始的 $\mathbf{x}, \mathbf{y}$ 可以表示为 $s_k = \sum x_i y_{i+k}$。

[HNOI2017] 礼物

容易发现旋转操作只会影响差异值表达中的最后一项。

如果 $\mathbf{y}$ 旋转了 k 个位置，这一项用初始的 $\mathbf{x}, \mathbf{y}$ 可以表示为 $s_k = \sum x_i y_{i+k}$。

可以发现，把数列 $\mathbf{x}$ 翻转以后与 $\mathbf{y}$ 的循环卷积就是 $\mathbf{s}$。

因此用 FFT 计算卷积，由此得到 $\mathbf{s}$，在其中找一个最大值，结合适当选取的 c，就得到了答案。

卷积与字符串匹配

假设 x, y 是两个用正整数表示的字符，那么 $x = y$ 当且仅当 $(x - y)^2 = 0$。

卷积与字符串匹配

假设 x, y 是两个用正整数表示的字符，那么 $x = y$ 当且仅当 $(x - y)^2 = 0$。

若 $\mathbf{a} = (a_i)$ 与 $\mathbf{b} = (b_i)$ 是长度分别是 n, m 的两个字符串，那么 $\mathbf{b}$ 在 $\mathbf{a}$ 的第 k 个位置匹配当且仅当

$$\sum_{i=1}^{m}(a_{i+k} - b_i)^2 = \sum_{i=1}^{m} a_{i+k}^2 + \sum_{i=1}^{m} b_i^2 - 2\sum_{i=1}^{m} a_{i+k}b_i = 0$$

卷积与字符串匹配

假设 x, y 是两个用正整数表示的字符，那么 $x = y$ 当且仅当 $(x - y)^2 = 0$。

若 $\mathbf{a} = (a_i)$ 与 $\mathbf{b} = (b_i)$ 是长度分别是 n, m 的两个字符串，那么 $\mathbf{b}$ 在 $\mathbf{a}$ 的第 k 个位置匹配当且仅当

$$\sum_{i=1}^{m}(a_{i+k} - b_i)^2 = \sum_{i=1}^{m} a_{i+k}^2 + \sum_{i=1}^{m} b_i^2 - 2\sum_{i=1}^{m} a_{i+k}b_i = 0$$

注意到其中前两项容易计算，而第三项可以化为卷积，因此可以使用 FFT 计算。

含有通配符的字符串匹配

如果 **b** 中含有一种特殊的字符（通配符）可以与 **a** 的任意字符匹配，如何求 **b** 在 **a** 中的匹配？

含有通配符的字符串匹配

如果 **b** 中含有一种特殊的字符（通配符）可以与 **a** 的任意字符匹配，如何求 **b** 在 **a** 中的匹配？

不妨把通配符用 0 表示，如果 x 是普通字符，y 可能是通配符，那么 x 与 y 匹配当且仅当 $x = y$ 或 $y = 0$，也就是 $y(x-y)^2 = 0$。

含有通配符的字符串匹配

如果 **b** 中含有一种特殊的字符（通配符）可以与 **a** 的任意字符匹配，如何求 **b** 在 **a** 中的匹配？

不妨把通配符用 0 表示，如果 x 是普通字符，y 可能是通配符，那么 x 与 y 匹配当且仅当 $x = y$ 或 $y = 0$，也就是 $y(x-y)^2 = 0$。

那么字符串的匹配就可以表达为：

$$\sum_{i=1}^{m} b_i(a_{i+k} - b_i)^2 = \sum_{i=1}^{m} a_{i+k}^2 b_i - 2\sum_{i=1}^{m} a_{i+k} b_i^2 + \sum_{i=1}^{m} b_i^3 = 0$$

类似的，这个转化后也可以利用 FFT 计算。

形式幂级数

环 R 上的形式幂级数形如

$$A(x) = \sum_{n \geq 0} a_n x^n$$

其中 $a_n \in R$。x 称为这个形式幂级数的自由元。

形式幂级数

环 R 上的形式幂级数形如

$$A(x) = \sum_{n \geq 0} a_n x^n$$

其中 $a_n \in R$。x 称为这个形式幂级数的自由元。

多项式是仅有有限项非零的形式幂级数。因此，形式幂级数可以看成是对多项式的推广。

形式幂级数

环 R 上的形式幂级数形如

$$A(x) = \sum_{n \geq 0} a_n x^n$$

其中 $a_n \in R$。x 称为这个形式幂级数的自由元。

多项式是仅有有限项非零的形式幂级数。因此，形式幂级数可以看成是对多项式的推广。

一般形式幂级数中的自由元不带入具体数值。

形式幂级数的运算

形式幂级数的运算规则与多项式的运算规则是类似的。

形式幂级数的运算

形式幂级数的运算规则与多项式的运算规则是类似的。

同样多项式类似，形式幂级数的乘法对应两个无穷数列的卷积。

生成函数

可以看出，形式幂级数 $A(x)$ 自然的对应着一个无穷的数列 $\mathbf{a} = (a_0, a_1, a_2, \dots)$。

生成函数

可以看出，形式幂级数 $A(x)$ 自然的对应着一个无穷的数列 $\mathbf{a} = (a_0, a_1, a_2, \dots)$。

因此 $A(x)$ 称为数列 $\mathbf{a}$ 的生成函数。

模意义下的多项式

设 $A(x), B(x), P(x)$ 是多项式。我们称 $A(x) \equiv B(x) \pmod{R(x)}$，如果存在多项式 $Q(x)$，使得

$$A(x) - B(x) = P(x)Q(x)$$

因此，多项式也可以在模意义下讨论。

在实际运算中，我们时常只关心多项式（形式幂级数）的前有限项，并且希望多项式（形式幂级数）在这种意义下参与运算。如果只关心前 n 项，我们采用记号 $\bmod x^n$ 表示。

形式幂级数的逆元

设 $A(x)$ 是形式幂级数。如果 $a_0 \neq 0$, 那么存在形式幂级数 $B(x)$，使得 $A(x)B(x) = 1$. 这时，称 $B(x)$ 是 $A(x)$ 的逆元，记作 $B(x) = 1/A(x)$。

形式幂级数的逆元

设 $A(x)$ 是形式幂级数。如果 $a_0 \neq 0$, 那么存在形式幂级数 $B(x)$，使得 $A(x)B(x) = 1$. 这时，称 $B(x)$ 是 $A(x)$ 的逆元，记作 $B(x) = 1/A(x)$。

通过比较系数，我们可以得到：

$$\sum_{i=0}^{k} a_i b_{k-i} = [k = 0] \implies \begin{cases} b_0 & = \frac{1}{a_0} \\ b_n & = -\frac{1}{a_0} \sum_{i=0}^{n-1} a_{n-i} b_i \end{cases}$$

形式幂级数的除法

通过定义逆元，我们可以定义除以一个形式幂级数为乘以其逆元。

斐波那契数列

定义斐波那契数列满足 $f_0 = f_1 = 1, f_n = f_{n-1} + f_{n-2}$。

求斐波那契数列的生成函数。

斐波那契数列

定义斐波那契数列满足 $f_0 = f_1 = 1, f_n = f_{n-1} + f_{n-2}$。

求斐波那契数列的生成函数。

可以发现我们有 $F(x) = (x + x^2)F(x) + 1$。因此答案是

$$F(x) = \frac{1}{1 - x - x^2}$$

逆元的计算

假设我们已经计算出了 $B_0(x) = B(x) \bmod x^k$，那么我们有

$$A(x)B_0(x) \equiv 1 \pmod{x^k}$$

逆元的计算

假设我们已经计算出了 $B_0(x) = B(x) \bmod x^k$，那么我们有

$$A(x)B_0(x) \equiv 1 \pmod{x^k}$$

这意味着

$$(A(x)B_0(x) - 1)^2 \equiv 0 \pmod{x^{2k}}$$

那么我们有

$$B(x) \equiv B_0(x)(2 - A(x)B_0(x)) \pmod{x^{2k}}$$

由此可以计算出 $g(x)$ 的前 $2k$ 项。

根据主定理，计算 $B(x) \bmod x^n$ 的时间复杂度为 $O(n \log n)$。

导数与积分

设 $[A(x) = \sum_{n\geq 0} a_n x^n$ 是形式幂级数。定义 $A(x)$ 的（形式）导数为

$$A'(x) = \frac{\mathrm{d}}{\mathrm{d}x} A(x) = \sum_{n\geq 1} n a_n x^{n-1}$$

类似的，定义定义 $A(x)$ 的（形式）积分为

$$\int A(x)\,\mathrm{d}x = \sum_{n\geq 0} \frac{1}{n+1} a_n x^{n+1}$$

对数

（自然）对数可以由下式定义：

$$\log(1+x) = \sum_{n\geq 1}(-1)^{n-1}\frac{x^n}{n}$$

设 $A(x)$ 是形式幂级数。如果 $[x^0]A(x) = 1$, 则 $A(x)$ 的对数也可以类似定义：

$$\log A(x) = \sum_{n\geq 1}(-1)^{n-1}\frac{(A(x)-1)^n}{n}$$

对数的计算

我们有

$$\frac{\mathrm{d}}{\mathrm{d}x}\log A(x)=\frac{A'(x)}{A(x)}$$

因此，

$$\log A(x)=\int\frac{A'(x)}{A(x)}\,\mathrm{d}x$$

利用多项式求逆，我们可以在 $O(n\log n)$ 时间内计算 $\log A(x)$ 的前 n 项。

等幂和

给出 n 个数 $\alpha_1, \ldots, \alpha_n$ 定义 $p_k = \sum_{i=1}^{n} {\alpha_i}^k$。对于 $1 \le k \le n$，求 p_k。

$n \le 10^5$。

等幂和

注意到 $\sum_{k\geq 0}\alpha^k x^k = \frac{1}{1-\alpha x}$。因此

$$\begin{aligned}\sum_{k\geq 1} p_k x^{k-1} &= \sum_{k\geq 0}\sum_{i=1}^{n} {\alpha_i}^k x^{k-1}\\ &= \sum_{i=1}^{n}\sum_{k\geq 0} {\alpha_i}^k x^{k-1}\\ &= \sum_{i=1}^{n}\frac{\alpha_i}{1-\alpha_i x}\end{aligned}$$

等幂和

又因为 $\int \frac{\alpha_i}{1-\alpha_i x}\,\mathrm{d}x = -\log(1-\alpha_i x)$。因此

$$\int \sum_{k\geq 0} p_k x^{k-1}\,\mathrm{d}x = \int \sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_i x}\,\mathrm{d}x$$

$$= \sum_{i=1}^{n} -\log(1-\alpha_i x) \qquad = -\log \prod_{i=1}^{n}(1-\alpha_i x)$$

$$= -\frac{\frac{\mathrm{d}}{\mathrm{d}x}\prod_{i=1}^{n}(1-\alpha_i x)}{\prod_{i=1}^{n}(1-\alpha_i x)}$$

分治计算 $\prod_{i=1}^{n}(1-\alpha_i x)$ 即可。

指数

指数函数可以由下式定义：

$$\exp(x) = \mathrm{e}^x = \sum_{n\geq 0} \frac{x^n}{n!}$$

设 $A(x)$ 是形式幂级数。如果 $[x^0]A(x) = 0$, 则 $A(x)$ 的指数也可以类似定义：

$$\exp(A(x)) = \sum_{n\geq 0} \frac{(A(x))^n}{n!}$$

指数的计算

设 $B(x) = \exp(A(x))$。假设我们已经计算出了 $B_0(x) = B(x) \bmod x^k$，那么有

$$B(x) \equiv B_0(x)(1 - \log B_0(x) + A(x)) \pmod{x^{2k}}$$

利用多项式求对数，我们可以倍增的在 $O(n \log n)$ 时间内计算 $\exp(A(x))$ 的前 n 项。

拓展阅读

- 毛啸，再探快速傅里叶变换，2016 年信息学奥林匹克中国国家队候选队员论文集
- 金策，生成函数的运算与组合计数问题，2015 年信息学奥林匹克中国国家队候选队员论文集

习题

- P3321 [SDOI2015] 序列统计
- P4705 玩游戏