

算法设计与分析

分治法 (II)

目录

- ① 快速幂
- ② 整数乘法
- ③ 矩阵乘法
- ④ 多项式乘法

快速幂问题

输入: $a \in \mathbb{R}$, $n \in \mathbb{N}$

输出: a^n

朴素算法: 顺序乘法

$$a^n = \underbrace{a \cdot a \cdot \dots \cdot a \cdot a}_n$$

#(乘法) = $n - 1$

分治法：分解

n 是偶数

$$\underbrace{a \cdot \dots \cdot a}_{n/2} \cdot \underbrace{a \cdot \dots \cdot a}_{n/2}$$

n 是奇数

$$\underbrace{a \cdot \dots \cdot a}_{(n-1)/2} \cdot \underbrace{a \cdot \dots \cdot a}_{(n-1)/2} \cdot a$$

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & n \text{ 是偶数} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & n \text{ 是奇数} \end{cases}$$

复杂度分析

基本操作：乘法

- 子问题规模：小于 $n/2$
- 两个子问题（规模约为 $n/2$ ）是相同的，只需计算一次

$$W(n) = W(n/2) + \Theta(1)$$

主定理（情况 1） $\Rightarrow W(n) = \Theta(\log n)$

如何实现这个算法？递归 vs. 迭代

递归方法

```
1: if  $n < 0$  then  
2:   return Power( $1/a, -n$ ) {处理负整数指数}  
3: end if  
4: if  $n = 0$  then  
5:   return 1  
6: end if  
7: if  $n = 1$  then  
8:   return  $a$   
9: end if  
10: if  $n$  是偶数 then  
11:   return Power( $a^2, n/2$ )  
12: end if  
13: if  $n$  是奇数 then  
14:   return  $a \times$  Power( $a^2, (n-1)/2$ )  
15: end if
```

朴素实现： $a^n = \text{Power}(a, n/2) \times a^n$

迭代方法

$$y = a^n = a^{\sum_{i=0}^k b_i 2^i} = \prod_{i=0}^k (a^{2^i})^{b_i}$$

```
1:  $(b_k, b_{k-1}, \dots, b_1, b_0) \leftarrow \text{BinaryDecomposition}(n)$ 
2:  $y \leftarrow 1$ 
3:  $\text{power} \leftarrow a$ 
4: for  $i = 0$  to  $k$  do
5:   if  $b_i = 1$  then
6:      $y \leftarrow y \times \text{power}$  {加}
7:   end if
8:    $\text{power} \leftarrow \text{power} \times \text{power}$  {倍增}
9: end for
10: return  $y$ 
```

也称为二进制幂算法

自然扩展到加法半群：倍增-加法 (double-and-add)

快速幂算法的应用

Fibonacci 数列：1, 1, 2, 3, 5, 8, 13, 21, ...

加入 $F_0 = 0$, 得到:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

问题：给定初始值 $F_0 = 0$, $F_1 = 1$, 计算 F_n

朴素算法：从 F_0, F_1, \dots 开始，重复计算

$$F_n = F_{n-1} + F_{n-2}$$

复杂度：顺序加法： $\Theta(n)$

Fibonacci 数列的性质

更好的算法？如何推导通项公式？

$$F_n = F_{n-1} + F_{n-2}$$

观察： F_n 是 F_{n-1} 和 F_{n-2} 的线性组合。这提示我们用线性代数来表示递推关系。

Fibonacci 数列的性质

更好的算法？如何推导通项公式？

$$F_n = F_{n-1} + F_{n-2}$$

观察： F_n 是 F_{n-1} 和 F_{n-2} 的线性组合。这提示我们用线性代数来表示递推关系。

命题：设 $\{F_n\}$ 是 Fibonacci 数列，则

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

用数学归纳法证明

基础： $n = 1$ ：

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

证明（归纳步骤）

假设对任意 n , 公式成立, 即:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

则对于 $n+1$, 根据 Fibonacci 数列的定义:

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

归纳前提 $\Rightarrow \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$

通过快速幂改进的算法

设

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

使用广义快速幂算法计算 M^n

时间复杂度

- 矩阵乘法次数 $T(n) = \Theta(\log n)$
- 每次矩阵乘法需要 8 次数乘法
- 总体复杂度是 $\Theta(\log n)$

通过快速幂改进的算法

设

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

使用广义快速幂算法计算 M^n

时间复杂度

- 矩阵乘法次数 $T(n) = \Theta(\log n)$
- 每次矩阵乘法需要 8 次数乘法
- 总体复杂度是 $\Theta(\log n)$

进一步改进

- M 可以对角化 ($M = PM'P^{-1}$) \Rightarrow 可以直接使用快速幂算法以获得更好的基本计算步骤 (矩阵乘法) 效率

目录

1 快速幂

2 整数乘法

3 矩阵乘法

4 多项式乘法

整数加法

加法：给定两个 n 位整数 a 和 b ，计算 $a + b$ 。

减法：给定两个 n 位整数 a 和 b ，计算 $a - b$ 。

小学算法： $\Theta(n)$ 位操作。

注：小学加减法算法是渐近最优的。

整数乘法

乘法：给定两个 n 位整数 a 和 b ，计算 $a \times b$ 。

小学方法： $\Theta(n^2)$ 位操作

$$\Theta(n^2) \text{ 原子位乘法} + \Theta(n^2) \text{ 原子位加法}$$

分治法：第一次尝试 (1/2)

分解：将两个 n 位整数 x 和 y 分成左右两半（低位和高位）。设 $m = n/2$ 。

$$x = 2^{n/2}x_L + x_R, \quad y = 2^{n/2}y_L + y_R$$

使用位移计算

$$x_L = \lfloor x/2^m \rfloor, \quad x_R = x \bmod 2^m$$

$$y_L = \lfloor y/2^m \rfloor, \quad y_R = y \bmod 2^m$$

例： $x = \underbrace{1011}_{x_L} \underbrace{0110}_{x_R} = 1011 \times 2^4 + 0110$

分治法：第一次尝试 (2/2)

$$\begin{aligned} xy &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

求解：递归地乘四个 $n/2$ 位整数（主要操作）

合并：加法和移位得到结果

$$T(n) = \underbrace{4 T(n/2)}_{\text{递归调用}} + \underbrace{\Theta(n)}_{\text{加法、移位}}$$

$$\text{主定理 (情况 1)} \Rightarrow T(n) = \Theta(n^2)$$

子问题数太多 → 与传统小学方法运行时间相同，效率没有提升。

如何加速这个方法？

Gauss 的技巧

Gauss 曾注意到，虽然两个复数的乘积

$$(a + bi)(c + di) = ac - bd + (bc + ad)i$$

似乎需要 4 次乘法，但实际上可以用 3 次完成：

$$bc + ad = (a + b)(c + d) - ac - bd$$

[Carl Friedrich Gauss 图片]

Karatsuba 算法

1960 年，Kolmogorov 在一次研讨会上猜想小学乘法算法是最优的。一周内，当时 23 岁的学生 Karatsuba 发现了一个更好的算法，从而反驳了这个猜想。Kolmogorov 对这一发现非常兴奋，并于 1962 年发表了论文。

[Anatolii Karatsuba 图片]

Karatsuba 算法：第一个渐近快于二次“小学”算法的算法。

减少子问题数

思路：通过 Gauss 的技巧利用子问题之间的依赖关系

$$\underbrace{x_L y_R + x_R y_L}_{\text{中间项}} = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

```
1: if  $n = 1$  then  
2:   return  $x \times y$   
3: else  
4:    $m \leftarrow \lceil n/2 \rceil$   
5:    $x_L \leftarrow \lfloor x/2^m \rfloor$ ;  $x_R \leftarrow x \bmod 2^m$   
6:    $y_L \leftarrow \lfloor y/2^m \rfloor$ ;  $y_R \leftarrow y \bmod 2^m$   
7:    $e \leftarrow \text{KARATSUBA}(x_L, y_L, m)$   
8:    $f \leftarrow \text{KARATSUBA}(x_R, y_R, m)$   
9:    $g \leftarrow \text{KARATSUBA}(x_L + x_R, y_L + y_R, m)$   
10:  return  $2^{2m}e + 2^m(g - e - f) + f$   
11: end if
```

理论

复杂度分析：现在，递推关系为

$$\begin{cases} T(n) = 3T(n/2) + \Theta(n) \\ T(1) = 1 \end{cases} \Rightarrow T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585})$$

- 合并和分解代价 $f(n)$ 低 $\rightarrow h(n)$ 主导总体复杂度。从 4 到 3 的常数因子改进发生在递归的每一层，复合效应导致显著更低的界。

[Toom-Cook (1963)] Karatsuba 方法的更快推广

[Schönhage-Strassen (1971)] 对于足够大的 n 更快

一些说明

实践说明：

- 通常不值得一直递归到 1 位。对于大多数处理器，16 位或 32 位乘法是单一操作。
- GNU 多精度库根据操作数大小使用不同算法。（用于 Maple、Mathematica、gcc、密码学……）

理论回顾（非正式）：

- 小学加减法算法是最优的，因为它们已经是本地的
- 小学乘法算法不是很优的，因为它不是很本地（全局相关）：分治有助于缩小局部性

目录

1 快速幂

2 整数乘法

3 矩阵乘法

4 多项式乘法

内积

内积：给定两个长度为 n 的向量 $a = (a_1, \dots, a_n)$ 和 $b = (b_1, \dots, b_n)$, 计算

$$c = \langle a, b \rangle = \sum_{i=1}^n a_i b_i$$

小学方法： $\Theta(n)$ 算术操作。

注：小学点积算法是渐近最优的。

矩阵乘法

矩阵乘法：给定两个 $n \times n$ 矩阵 X 和 Y , 计算

$$Z = XY, \quad Z_{ij} = \sum_{k=1}^n X_{ik} Y_{kj}$$

大学方法： $\Theta(n^3)$ 算术操作

- Z 中有 n^2 个元素
- 计算每个元素需要 n 次算术乘法
- 大学矩阵乘法算法是渐近最优的吗？分治策略能做得更好吗？

朴素分治法

策略：将矩阵分成块：

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix}$$

其中：

$$Z_{11} = X_{11} Y_{11} + X_{12} Y_{21}$$

$$Z_{21} = X_{21} Y_{11} + X_{22} Y_{21}$$

$$Z_{12} = X_{11} Y_{12} + X_{12} Y_{22}$$

$$Z_{22} = X_{21} Y_{12} + X_{22} Y_{22}$$

递推关系：主定理（情况 1）

$$T(n) = \underbrace{8 T(n/2)}_{\text{递归调用}} + \underbrace{\Theta(n^2)}_{\text{加法/形成子矩阵}} \Rightarrow T(n) = \Theta(n^3)$$

突破

大学算法: $\Theta(n^3)$

朴素分治策略: $\Theta(n^3)$ (不令人印象深刻)

- 在相当长一段时间里, 人们普遍认为这是可能的最佳运行时间, 甚至有人证明在某些模型中没有算法能做得更好。

令人兴奋的是: 通过一些巧妙的代数, 这个效率可以进一步提高。

Strassen 算法 (1/3)

Volker Strassen 于 1969 年首次发表这个算法

- 证明了 $\Theta(n^3)$ 的一般矩阵乘法算法不是最优的
- 比标准矩阵乘法算法更快，对于大矩阵在实践中有用
- 启发了更多关于矩阵乘法的研究，导致了更快的方法，如 Coppersmith-Winograd 算法

[Volker Strassen 图片]

Strassen 算法 (2/3)

定义 7 个中间矩阵：

$$M_1 = X_{11}(Y_{12} - Y_{22})$$

$$M_2 = (X_{11} + X_{12})Y_{22}$$

$$M_3 = (X_{21} + X_{22})Y_{11}$$

$$M_4 = X_{22}(Y_{21} - Y_{11})$$

$$M_5 = (X_{11} + X_{22})(Y_{11} + Y_{22})$$

$$M_6 = (X_{12} - X_{22})(Y_{21} + Y_{22})$$

$$M_7 = (X_{11} - X_{21})(Y_{11} + Y_{12})$$

用中间矩阵表示 Z_{ij} ：

$$Z_{11} = M_5 + M_4 - M_2 + M_6$$

$$Z_{21} = M_3 + M_4$$

$$Z_{12} = M_1 + M_2$$

$$Z_{22} = M_5 + M_1 - M_3 - M_7$$

$$Z_{12} = M_1 + M_2 = X_{11}(Y_{12} - Y_{22}) + (X_{11} + X_{12})Y_{22} = X_{11}Y_{12} + X_{12}Y_{22}$$

Strassen 算法 (3/3)

将子问题数从 8 减少到 7

时间复杂度的递推关系 (18 是每次应用算法执行的加/减法次数)

$$\begin{cases} T(n) = 7T(n/2) + 18n^2 \\ T(1) = 1 \end{cases} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.8075})$$

Strassen 算法 (3/3)

将子问题数从 8 减少到 7

时间复杂度的递推关系 (18 是每次应用算法执行的加/减法次数)

$$\begin{cases} T(n) = 7T(n/2) + 18n^2 \\ T(1) = 1 \end{cases} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.8075})$$

问：如果 n 不是 2 的幂怎么办？

答：可以用零填充矩阵。

关于矩阵乘法的更多内容

这个分解如此巧妙复杂，以至于人们不禁想知道 Strassen 是如何发现它的！

关于矩阵乘法的更多内容

这个分解如此巧妙复杂，以至于人们不禁想知道 Strassen 是如何发现它的！

矩阵乘法的复杂度

- 最佳上界： $O(n^{2.376})$ —Coppersmith-Winograd 算法
- 已知下界： $\Omega(n^2)$

关于矩阵乘法的更多内容

这个分解如此巧妙复杂，以至于人们不禁想知道 Strassen 是如何发现它的！

矩阵乘法的复杂度

- 最佳上界： $O(n^{2.376})$ —Coppersmith-Winograd 算法
- 已知下界： $\Omega(n^2)$

应用

- 科学计算、图像处理、数据挖掘（回归、聚合、决策树）

目录

1 快速幂

2 整数乘法

3 矩阵乘法

4 多项式乘法

动机

我们已经学习了如何相乘

- 整数: Gauss 的技巧
- 矩阵: Strassen 算法

动机

我们已经学习了如何相乘

- 整数: Gauss 的技巧
- 矩阵: Strassen 算法

如何相乘多项式?

动机

我们已经学习了如何相乘

- 整数: Gauss 的技巧
- 矩阵: Strassen 算法

如何相乘多项式?

多项式乘法的应用

- 最快的多项式乘法意味着最快的整数乘法
 - ▶ 多项式和二进制整数非常相似——只需将变量 x 替换为基数 2 并注意进位
- 多项式相乘对信号处理至关重要

多项式：系数表示

多项式 [系数表示]

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$$

多项式运算

加法: $\Theta(n)$

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

多项式运算

加法: $\Theta(n)$

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

求值: 三种选择

- 朴素算法: 逐项计算: $\Theta(n^2)$
- 缓存算法: 缓存 x^i : $\Theta(n)$
- Horner 算法: $a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))))): \Theta(n)$
- 秦九韶早在几百年前就发现了这个算法

多项式运算

加法: $\Theta(n)$

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

求值: 三种选择

- 朴素算法: 逐项计算: $\Theta(n^2)$
- 缓存算法: 缓存 x^i : $\Theta(n)$
- Horner 算法: $a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))))): \Theta(n)$
- 秦九韶早在几百年前就发现了这个算法

乘法 (卷积): 使用暴力算法 $\Theta(n^2)$

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ 其中 } c_i = \sum_{j=0}^i a_j b_{i-j}$$

卷积的图示

$$(c_0, \dots, c_{2n-2}) = (a_0, \dots, a_{n-1}) \circledast (b_0, \dots, b_{n-1})$$

多项式：点值表示

代数基本定理 [Gauss 博士论文]

每个具有复系数的非零单变量 n 次多项式恰好有 n 个复根。

推论： $n - 1$ 次多项式 $A(x)$ 由其在 n 个不同点的求值唯一确定。

- 假设另一个 $n - 1$ 次多项式 $A'(x)$ 在 n 个不同点与 $A(x)$ 有相同的求值
- $\Rightarrow A(x) - A'(x)$ 最多是 $n - 1$ 次但有 n 个根 \rightarrow 与代数基本定理矛盾

多项式：点值表示

多项式 [点值表示]

$$A(x) : (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

$$B(x) : (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

多项式运算

加法: $\Theta(n)$ 次加法操作

$$A(x) + B(x) : (x_0, y_0 + z_0), \dots, (x_{n-1}, y_{n-1} + z_{n-1})$$

乘法 (卷积): $\Theta(n)$, 但需要 $2n - 1$ 个点

$$A(x) \times B(x) : (x_0, y_0 \times z_0), \dots, (x_{2n-2}, y_{2n-2} \times z_{2n-2})$$

求值: 使用 Lagrange 公式 $\Theta(n^2)$

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

两种表示之间的转换

权衡：快速求值还是快速乘法

表示	乘法	求值
系数	$\Theta(n^2)$	$\Theta(n)$
点值	$\Theta(n)$	$\Theta(n^2)$

目标：两种表示之间的高效转换 \Rightarrow 享受两个世界的最好：所有操作都快！

两种表示之间的转换：求值

系数 \Rightarrow 点值

给定 $A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$, 在 n 个不同点 x_0, \dots, x_{n-1} 处求值。

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

运行时间：矩阵-向量乘法（或 n 次 Horner） $\Theta(n^2)$

两种表示之间的转换：插值

点值 \Rightarrow 系数

给定 n 个不同点 x_0, \dots, x_{n-1} 和值 (y_0, \dots, y_{n-1}) , 找唯一多项式
 $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, 在给定点有给定值。

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & \dots & x_0^{n-1} \\ 1 & x_1 & \dots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \dots & x_{n-1}^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Vandermonde 矩阵可逆当且仅当 x_i 互不相同。

运行时间：高斯消元 $\Theta(n^3)$

重述目标

两种已知转换都不高效

- 系数 \Rightarrow 点值: $\Theta(n^2)$
- 点值 \Rightarrow 系数: $\Theta(n^3)$

需要更高效的转换。

重述目标

两种已知转换都不高效

- 系数 \Rightarrow 点值: $\Theta(n^2)$
- 点值 \Rightarrow 系数: $\Theta(n^3)$

需要更高效的转换。

接下来, 我们从第一个方向开始。重述目标:

给定 n 个系数, 快速计算 n 个点值元组。

重述目标

两种已知转换都不高效

- 系数 \Rightarrow 点值: $\Theta(n^2)$
- 点值 \Rightarrow 系数: $\Theta(n^3)$

需要更高效的转换。

接下来, 我们从第一个方向开始。重述目标:

给定 n 个系数, 快速计算 n 个点值元组。

多项式求值算法的最优复杂度是 $\Theta(n)$ 。因此, 上述目标的 $\Theta(n^2)$ 复杂度似乎不可避免。

重述目标

两种已知转换都不高效

- 系数 \Rightarrow 点值: $\Theta(n^2)$
- 点值 \Rightarrow 系数: $\Theta(n^3)$

需要更高效的转换。

接下来, 我们从第一个方向开始。重述目标:

给定 n 个系数, 快速计算 n 个点值元组。

多项式求值算法的最优复杂度是 $\Theta(n)$ 。因此, 上述目标的 $\Theta(n^2)$ 复杂度似乎不可避免。
高层关键思路: 施加结构以增强局部性 \rightarrow 降低复杂度

求值的分治法

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$$

分解的两种选择：频率 vs. 时间

频率抽取：按低次和高次幂分解

$$A_{\text{low}}(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

$$A_{\text{high}}(x) = a_4 + a_5x + a_6x^2 + a_7x^3$$

$$A(x) = A_{\text{low}}(x) + x^4 A_{\text{high}}(x)$$

时间抽取：按偶次和奇次幂分解

$$A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + a_6x^3$$

$$A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + a_7x^3$$

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

基 2 时间抽取 (DIT)

澄清

我们强调，分解的目标不是提高单点多项式求值的效率，因为它已经是最优的。
最终目标是提高作为整体任务的 n 个点求值的效率。

第一次尝试

朴素想法：随机选取 n 个不同点 x_0, \dots, x_{n-1} ，然后通过 $A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$ 计算 $A(x)$ 。

- $T(n)$: 在 n 个点求值 $n - 1$ 次多项式
- $E(n)$: 在 1 个点求值 $n - 1$ 次多项式

第一次尝试

朴素想法：随机选取 n 个不同点 x_0, \dots, x_{n-1} ，然后通过 $A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$ 计算 $A(x)$ 。

- $T(n)$: 在 n 个点求值 $n - 1$ 次多项式
- $E(n)$: 在 1 个点求值 $n - 1$ 次多项式

问题：效率没有提高

- 在 n 个点求值 $n - 1$ 次 $A(x)$: $T(n) = n \cdot E(n)$
- 在 n 个点求值 $n/2 - 1$ 次的 $A_{\text{even}}(x)$ 和 $A_{\text{odd}}(x)$: $2 \times n \cdot E(n/2) = 2n \cdot E(n/2)$

$E(n)$ 是线性函数 \rightarrow 效率没有提高

- 根源是问题规模没有完全减半

解决方案：减少求值点的数量

基本思路 (1/2)

基本思路：通过选择 n 个点为正负配对来引入简单结构，即

$$\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$$

注意 x_i 的偶次幂与 $-x_i$ 的一致 \Rightarrow 每个 $A(x_i)$ 和 $A(-x_i)$ 所需的计算大量重叠。

$$A(x_i) = A_{\text{even}}(x_i^2) + x_i A_{\text{odd}}(x_i^2)$$

$$A(-x_i) = A_{\text{even}}(x_i^2) - x_i A_{\text{odd}}(x_i^2)$$

现在，在 n 个配对点 $\pm x_0, \dots, \pm x_{n/2-1}$ 处求值 $n-1$ 次多项式 $A(x) \Rightarrow$ 在仅 $n/2$ 个点 $x_0^2, \dots, x_{n/2-1}^2$ 处求值 $n/2-1$ 次多项式 $A_{\text{even}}(x)$ 和 $A_{\text{odd}}(x)$ 。

基本思路 (2/2)

现在，规模为 n 的原问题以这种方式被转化为两个规模为 $n/2$ 的子问题，后跟一些线性时间算术。

- $T(n)$: 在 n 个点求值 $n - 1$ 次多项式

如果我们能递归，我们将得到一个运行时间为：

$$T(n) = 2 T(n/2) + \Theta(n)$$

的分治过程，即 $\Theta(n \log n)$ ，正是我们想要的。

技术障碍

技术障碍：正负配对技巧只在递归的顶层有效。

- 要在下一层递归，我们需要 $n/2$ 个求值点 $x_0^2, x_1^2, \dots, x_{n/2-1}^2$ 本身是正负配对的。

技术障碍

技术障碍：正负配对技巧只在递归的顶层有效。

- 要在下一层递归，我们需要 $n/2$ 个求值点 $x_0^2, x_1^2, \dots, x_{n/2-1}^2$ 本身是正负配对的。

但平方怎么能是负的？

技术障碍

技术障碍：正负配对技巧只在递归的顶层有效。

- 要在下一层递归，我们需要 $n/2$ 个求值点 $x_0^2, x_1^2, \dots, x_{n/2-1}^2$ 本身是正负配对的。

但平方怎么能是负的？

除非，当然，我们使用复数。

哪些复数？

好吧，但是哪些复数？让我们通过“逆向工程”这个过程来搞清楚。

- 在递归底部，我们有一个点，比如 1。
- 在它上面的层必须由它的平方根组成， ± 1 。
- 再上一层是 $(+1, -1)$ 和 $(+i, -i)$ ，直到达到 $n = 2^k$ 个叶节点。

n 个复数的选择

n 次单位根是满足 $x^n = 1$ 的复数。

事实： n 次单位根是 $\omega^0 = 1, \omega^1, \omega^2, \dots, \omega^{n-1}$ ，其中

$$\omega = e^{2\pi i/n} = \cos \frac{2\pi}{n} + i \sin \frac{2\pi}{n}$$

证明： $(\omega^k)^n = (e^{2\pi ik/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$

$$e^{ix} = \cos x + i \sin x$$

n 个复数的选择

n 次单位根是满足 $x^n = 1$ 的复数。

事实： n 次单位根是 $\omega^0 = 1, \omega^1, \omega^2, \dots, \omega^{n-1}$ ，其中

$$\omega = e^{2\pi i/n} = \cos \frac{2\pi}{n} + i \sin \frac{2\pi}{n}$$

证明： $(\omega^k)^n = (e^{2\pi ik/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$

$$e^{ix} = \cos x + i \sin x$$

如果 n 是偶数， n 次单位根是正负配对的， $\omega^{n/2+j} = -\omega^j$

- 对它们平方产生 $(n/2)$ 次单位根： $v^0, v^1, \dots, v^{n/2-1}$ ，其中 $v = \omega^2 = e^{4\pi i/n}$

n 个复数的选择

n 次单位根是满足 $x^n = 1$ 的复数。

事实： n 次单位根是 $\omega^0 = 1, \omega^1, \omega^2, \dots, \omega^{n-1}$, 其中

$$\omega = e^{2\pi i/n} = \cos \frac{2\pi}{n} + i \sin \frac{2\pi}{n}$$

证明： $(\omega^k)^n = (e^{2\pi ik/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$

$$e^{ix} = \cos x + i \sin x$$

如果 n 是偶数, n 次单位根是正负配对的, $\omega^{n/2+j} = -\omega^j$

- 对它们平方产生 $(n/2)$ 次单位根: $v^0, v^1, \dots, v^{n/2-1}$, 其中 $v = \omega^2 = e^{4\pi i/n}$

如果我们从某个 $n = 2^k$ 的 $\omega^0, \omega^1, \omega^2, \dots, \omega^{n-1}$ 开始, 那么在 k 层递归时我们将有 $(n/2^k)$ 次单位根。

- 所有这些单位根集合都是正负配对的 \Rightarrow 分治算法将完美工作

$n = 8$ 的演示

$$\omega^0 = 1$$

$$\omega^1 = e^{\pi i/4} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i$$

$$\omega^2 = e^{\pi i/2} = i$$

$$\omega^3 = e^{3\pi i/4} = -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i$$

$$\omega^4 = e^{\pi i} = -1$$

$$\omega^5 = -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$$

$$\omega^6 = -i$$

$$\omega^7 = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$$

递归结构与 FFT

DFT: Fourier 矩阵 $M_n(\omega)$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 2} & \dots & \omega^{0\cdot(n-1)} \\ 1 & \omega^1 & \omega^{1\cdot 2} & \dots & \omega^{1\cdot(n-1)} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \dots & \omega^{2\cdot(n-1)} \\ 1 & \omega^3 & \omega^{3\cdot 2} & \dots & \omega^{3\cdot(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)\cdot 2} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

快速 Fourier 变换 (FFT)

细化目标：在其 n 次单位根 $\omega^0, \omega^1, \dots, \omega^{n-1}$ 处求值 $A(x) = a_0 + \dots + a_{n-1}x^{n-1}$

分解：按偶次和奇次幂分解多项式：

$$A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$$

$$A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$$

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

求解：在 $(n/2)$ 次单位根 $v^0, v^1, \dots, v^{n/2-1}$ 处求值 $A_{\text{even}}(x)$ 和 $A_{\text{odd}}(x)$

$$v^k = (\omega^k)^2$$

合并：

$$A(\omega^k) = A_{\text{even}}(v^k) + \omega^k A_{\text{odd}}(v^k), \quad 1 \leq k < n/2$$

$$A(\omega^{k+n/2}) = A_{\text{even}}(v^k) - \omega^k A_{\text{odd}}(v^k), \quad 1 \leq k < n/2$$

FFT 算法的伪代码

Require: $n - 1$ 次多项式 A 的系数表示，主 n 次单位根 $\omega = e^{2\pi i/n}$

Ensure: 值表示 $A(\omega^0), \dots, A(\omega^{n-1})$

```
1: if  $n = 1$  then
2:   return  $a_0$ 
3: end if
4: 表示  $A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$ 
5: FFT( $A_{\text{even}}, n/2, \omega^2$ )  $\rightarrow (A_{\text{even}}((\omega^2)^0), \dots, A_{\text{even}}((\omega^2)^{n/2-1}))$ 
6: FFT( $A_{\text{odd}}, n/2, \omega^2$ )  $\rightarrow (A_{\text{odd}}((\omega^2)^0), \dots, A_{\text{odd}}((\omega^2)^{n/2-1}))$ 
7: for  $j = 0$  to  $n - 1$  do
8:    $A(\omega^j) = A_{\text{even}}(\omega^{2j}) + \omega^j A_{\text{odd}}(\omega^{2j}) \{\Theta(n)\}$ 
9: end for
10: return  $A(\omega^0), \dots, A(\omega^{n-1})$ 
```

FFT 小结

定理：假设 $n = 2^k$ 。FFT 算法在 $\Theta(n \log n)$ 步内在每个 n 次单位根处求值 $n - 1$ 次多项式。
运行时间

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

本质：选择具有特殊结构的 n 个点来加速 DFT 计算。

$$a_0, a_1, \dots, a_{n-1} \xrightarrow{\Theta(n \log n)} (\omega_0, y_0), \dots, (\omega_{n-1}, y_{n-1})$$

系数表示 $\xrightarrow{???$ 点值表示}

回顾

我们首先开发了一种高层多项式乘法方法

系数表示 \Rightarrow 点值表示

点值表示使多项式乘法变得简单，但算法的输入输出形式指定为系数表示。

- 所以我们设计了 FFT：系数 \Rightarrow 点值，时间仅 $\Theta(n \log n)$ ，其中点 $\{x_i\}_n$ 是复 n 次单位根 $(1, \omega, \omega^2, \dots, \omega^{n-1})$ 。

$$\langle \text{values} \rangle = \text{FFT}(\langle \text{coefficients} \rangle, \omega)$$

插值

拼图的最后一块是逆运算——插值。令人惊奇的是：

$$\langle \text{coefficients} \rangle = \frac{1}{n} \text{FFT}(\langle \text{values} \rangle, \omega^{-1})$$

插值因此以最简单优雅的方式解决，使用相同的 FFT 算法，但用 ω^{-1} 代替 ω ！

- 这可能看起来像是一个奇迹般的巧合，但用线性代数的语言重新表述多项式运算时会更有意义。

关键事实

$$G_n(\omega) := \frac{1}{n} F_n(\omega^{-1}) = F_n(\omega)^{-1}$$

断言： F_n 和 G_n 互逆

证明：检验 $F_n G_n$

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{若 } k = k' \\ 0 & \text{否则} \end{cases}$$

求和引理：设 ω 是主 n 次单位根。则

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{若 } k = 0 \pmod{n} \\ 0 & \text{否则} \end{cases}$$

逆 FFT 小结

定理：假设 $n = 2^k$ 。逆 FFT 算法在 $\Theta(n \log n)$ 步内，给定在每个 n 次单位根处的值，插值出 $n - 1$ 次多项式。

运行时间：与 FFT 几乎相同的算法。

$$a_0, a_1, \dots, a_{n-1} \xrightarrow[\Theta(n \log n)]{\Theta(n \log n)} (\omega_0, y_0), \dots, (\omega_{n-1}, y_{n-1})$$

系数表示 \longleftrightarrow 点值表示

多项式乘法

定理：两个 $n - 1$ 次多项式可以在 $\Theta(n \log n)$ 步内相乘。（用 0 填充使 n 成为 2 的幂）

- 2 次 FFT: $\Theta(n \log n)$
- 点值乘法: $\Theta(n)$
- 1 次逆 FFT: $\Theta(n \log n)$

实际上， $2n - 1$ 个点值元组就足够了。但 FFT 要求输入大小为 2^k ，输出大小也是如此。

FFT 的扩展

FFT 在复数域 \mathbb{C} 中工作，根可能是复数 \rightarrow 精度损失不可避免
有时我们只需在有限域中工作，如 $F = \mathbb{Z}/p$, 模素数 p 的整数。

- 当 n 整除 $p - 1$ 时存在主 n 次单位根，即 $p = \xi n + 1$ (ξ 是正整数)
- 特别地，设 ω 是主 $(p - 1)$ 次单位根，则 n 次单位根 α 可通过令 $\alpha = \omega^\xi$ 找到

扩展 \Rightarrow 数论变换 (NTT)：通过将离散 Fourier 变换特化到 F 得到。

- 无精度损失且更快
- 在 SNARK 实现中广泛使用，如 libsnark

FFT 的应用

- 光学、声学、量子物理、电信、雷达、控制系统、信号处理、语音识别、数据压缩、图像处理、地震学、质谱学……
- 数字媒体 [DVD, JPEG, MP3, H.264]
- 医学诊断 [MRI, CT, PET 扫描, 超声波]
- Poisson 方程的数值解
- Shor 的量子因式分解算法

Gilbert Strang 将 FFT 描述为“我们有生之年最重要的数值算法”。

Fourier 分析

Fourier 定理 [Fourier, Dirichlet, Riemann]: 任何（足够光滑的）周期函数都可以表示为一系列正弦函数之和。

Euler 恒等式:

$$e^{ix} = \cos x + i \sin x$$

正弦函数：正弦和余弦之和 = 复指数之和

Fourier 变换

FFT 的信号处理视角

在时域和频域之间快速转换

FFT 的算法视角

快速乘法和求值多项式

FFT：简史

Gauss: 分析小行星 Ceres 的周期运动（用拉丁文）

Runge-König (1924): 奠定理论基础

Danielson-Lanczos (1942): 高效算法, X 射线晶体学

Cooley-Tukey (1965): 监测苏联核试验和追踪潜艇。重新发现并推广了 FFT。

直到数字计算机出现，其重要性才被充分认识。

整数乘法，重访

整数乘法：给定两个 n 位整数 $a = a_{n-1} \dots a_1 a_0$ 和 $b = b_{n-1} \dots b_1 b_0$ ，计算它们的乘积 ab 。

- ① 形成两个多项式（基 2 表示 $\Rightarrow a = A(2)$, $b = B(2)$ ）

$$A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + \dots + b_{n-1} x^{n-1}$$

- ② 通过 FFT 计算 $C(x) = A(x)B(x)$ ，求值 $C(2) = ab$

运行时间： $\Theta(n \log n)$ 复数算术操作。

实践：GMP 根据 n 的大小使用暴力、Karatsuba 和 FFT。

小结 (1/3)

分治法的概念

主要思想：将问题归约为子问题

原则：子问题应与原问题类型相同，且可独立求解。

- 直接分解：将原问题分成规模大致相同的子问题
 - ▶ FindMinMax、归并排序
- 精巧分解
 - ▶ 一般选择：使用中位数的中位数作为基准（找基准本身需要努力）
 - ▶ 最近点对：中线周围带的分析
 - ▶ 凸包：有时难以平衡分割

小结 (2/3)

实现：递归或迭代（注意可以直接求解的最小子问题）

时间复杂度：找递推关系和初始值，求解递推关系

分治算法的递推关系

$$T(n) = aT(n/b) + f(n)$$

- a : 子问题数, n/b : 子问题规模
- $f(n)$: 分解和合并的代价

小结 (3/3)

优化技巧 1：减少子问题数：当 $f(n)$ 不是很大时， $h(n) = n^{\log_b a}$ 主导总体复杂度 $\Rightarrow T(n) = \Theta(h(n))$

- 减少 a 可立即降低 $T(n)$ 的阶
- 当子问题相关时 \rightarrow 利用关系通过组合其他子问题的解来解决某些子问题

例

- 快速幂算法：子问题相同
- 简单代数技巧：整数乘法 ($f(n)$ 仍然低)
- 利用依赖：矩阵乘法 ($f(n)$ 可能增加但不影响阶)

优化技巧 2：减少分解和合并代价 $f(n)$ ：添加全局预处理

- 最近点对

重要的分治算法

- 搜索算法：二分查找
- 排序算法：快速排序、归并排序
- 选择算法：找最大/最小、一般选择算法
- 最近点对、凸包
- 快速幂算法
- 矩阵相乘：Strassen 算法
- 整数、多项式相乘：FFT

从前慢

随着算法不断拨快生活的节奏，内心越希望可以偶尔慢下来，松下烹茶、雨夜听琴，享受一份从容安宁。

——我的心声

从前慢，慢的不仅是车、马与邮件，还有在等待中默默酝酿的心。一封薄薄的信，装着一份炽热的情，翻过山淌过水送至一生唯一的爱人手中，信上的折痕都是那么饱含爱意……

——网易云音乐评论