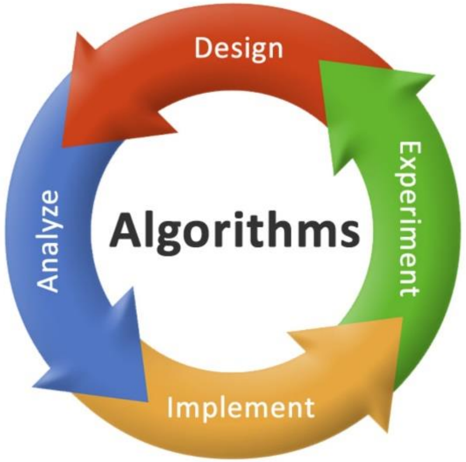


华中科技大学网络空间安全学院
School of Cyber Science and Engineering, HUST

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis

算法设计与分析

Chapter 5-2: Divide And Conquer II ▌ 张乾坤

课程提要

- 主定理（Master Theorem）
- 整数乘法（integer multiplication）
- 矩阵乘法（matrix multiplication）

课程提要

- 主定理（Master Theorem）
- 整数乘法（integer multiplication）
- 矩阵乘法（matrix multiplication）

分治递归

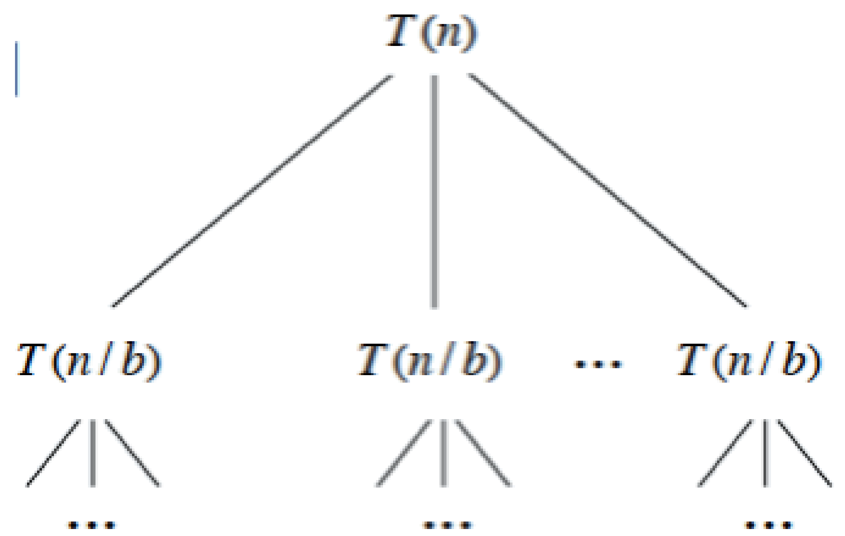
- 目标：解决常见分治递归问题的方法：

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中：

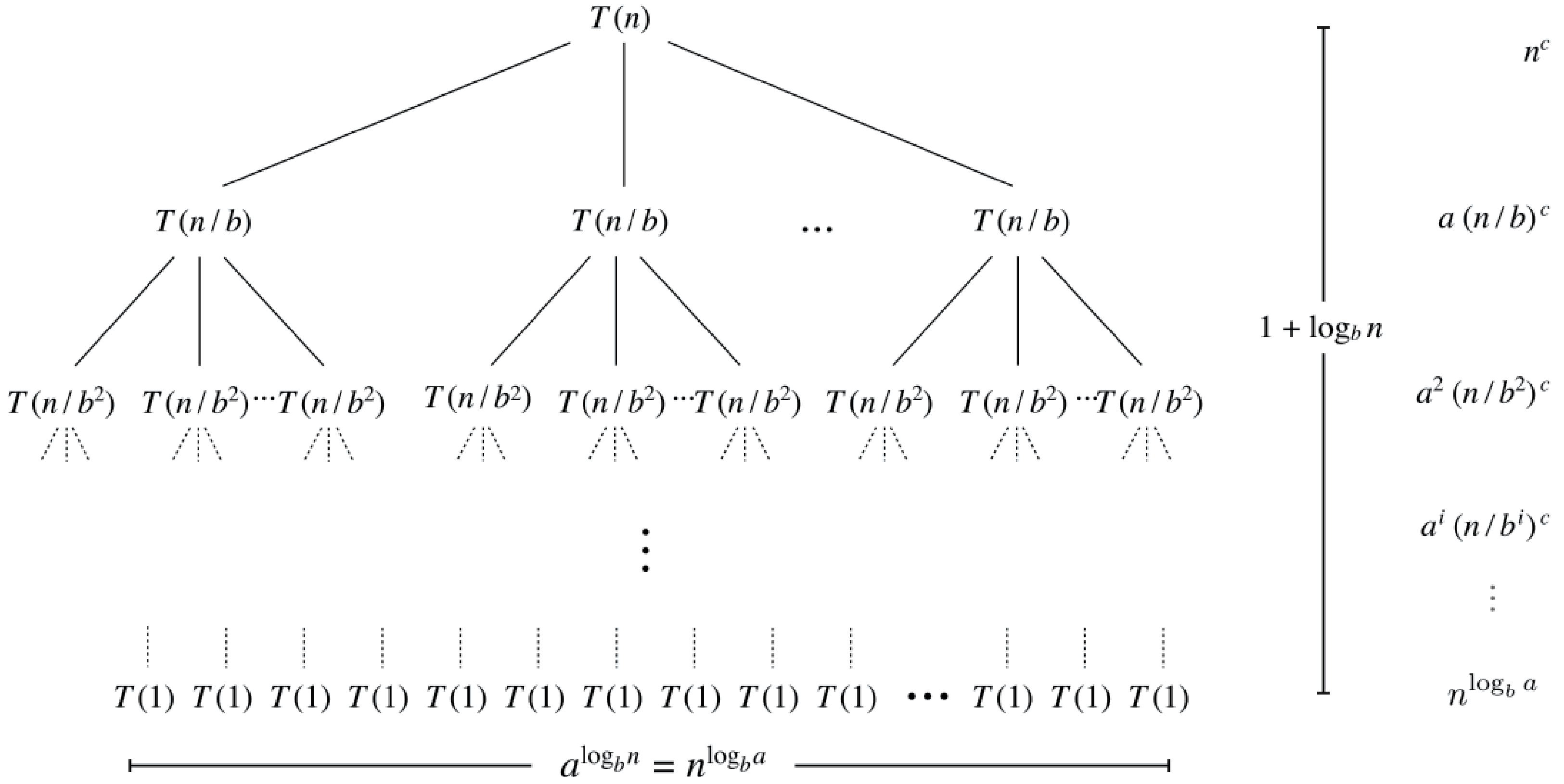
  - $a \geq 1$是子问题的数量；
  - $b \geq 2$是子问题大小减小的因子；
  - $f(n) \geq 0$是是划分和组合子问题的工作；

- 递归树（Recursion tree）[假设n是b的幂]
  - a = 分支系数；
  - a^i = 第i层子问题的数量；
  - $1 + \log_b n$层；
  - n/b^i = 第i级子问题的大小；



分治递归:递归树

- 假设$T(n)$满足$T(n) = aT(n/b) + n^c$且$T(1) = 1$， n是b的幂。



$r = a / b^c \qquad T(n) = n^c \sum_{i=0}^{\log_b n} r^i$

分治递归:递归树

- 假设$T(n)$满足$T(n) = aT(n/b) + n^c$且$T(1) = 1$， n是b的幂。
- 设$r = a/b^c$，注意到当且仅当$c > \log_b a$时$r < 1$。

$$T(n) = n^c \sum_{i=0}^{\log_b n} r^i = \begin{cases} \Theta(n^c) & \text{if } r < 1 \\ \Theta(n^c \log n) & \text{if } r = 1 \\ \Theta\left(n^{\log_b a}\right) & \text{if } r > 1 \end{cases}$$

$c > \log_b a \longleftarrow$ 成本由根的成本支配

$c = \log_b a \longleftarrow$ 成本均匀分布在树上

$c < \log_b a \longleftarrow$ 成本由叶子的成本支配

几何级数（Geometric series）

- 如果$0 < r < 1$，则$1 + r + r^2 + r^3 + \cdots + r^k \leq 1/(1-r)$。
- 如果$r = 1$， 则 $1 + r + r^2 + r^3 + \cdots + r^k = \mathrm{k} + 1$。
- 如果$r > 1$， 则 $1 + r + r^2 + r^3 + \cdots + r^k \leq (r^{k+1} - 1)/(r-1)$。

分治递归：主定理

主定理：设$a \geq 1$, $\mathrm{b} \geq 2$, $\mathrm{c} \geq 0$,设$T(n)$是一个关于非负整数的函数，满足递归式:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中n/b表示$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。则:

- Case 1: If $\mathrm{c} > \log_b a$, then $T(n) = \Theta(n^c)$.
- Case 2: If $\mathrm{c} = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
- Case 3: If $\mathrm{c} < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

分治递归：主定理

主定理：设$a \geq 1$, b ≥ 2, c ≥ 0,设$T(n)$是一个关于非负整数的函数，满足递归式:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中n/b表示$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。则:

- Case 1: If $\mathrm{c} > \log_b a$, then $T(n) = \Theta(n^c)$.
- Case 2: If $\mathrm{c} = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
- Case 3: If $\mathrm{c} < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

拓展:

- 可以用O替换Θ。
- 可以用Ω替换Θ。
- 对于所有$\mathrm{n} \leq n_0$，可以用$T(n) = \Theta(1)$替换初始条件。并且要求递归只适用于所有$\mathrm{n} > n_0$。

分治递归：主定理

主定理：设$a \geq 1$, b ≥ 2, c ≥ 0,设$T(n)$是一个关于非负整数的函数，满足递归式:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中n/b表示$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。则:

- Case 1: If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
- Case 2: If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
- Case 3: If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 1] $T(n) = 3T(\lfloor n/2 \rfloor) + 5n$

- $a = 3$, $b = 2$, $c = 1 < \log_b a = 1.5849 \ldots$.
- $T(n) = \Theta(n^{\log_3 2}) = O(n^{1.58})$.

分治递归：主定理

主定理：设$a \geq 1, \mathrm{b} \geq 2, \mathrm{c} \geq 0$,设$T(n)$是一个关于非负整数的函数，满足递归式:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中n/b表示$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。则:

- Case 1: If $\mathrm{c} > \log_b a$, then $T(n) = \Theta(n^c)$.
- Case 2: If $\mathrm{c} = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
- Case 3: If $\mathrm{c} < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 2] $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 17n$ ← 同时使用上下取整是可以的

- $\mathrm{a} = 2$, $\mathrm{b} = 2, \mathrm{c} = 1 = \log_b a$.
- $T(n) = \Theta\ (n\log n)$.

分治递归：主定理

主定理：设$a \ge 1$, b ≥ 2, c ≥ 0,设$T(n)$是一个关于非负整数的函数，满足递归式:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c)$$

$T(0) = 0$，$T(1) = \Theta(1)$。其中n/b表示$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。则:

- Case 1: If c > $\log_b a$, then $T(n) = \Theta(n^c)$.
- Case 2: If c = $\log_b a$, then $T(n) = \Theta(n^c \log n)$.
- Case 3: If c < $\log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 3] $T(n) = 48T(\lfloor n/4 \rfloor) + n^3$

- a = 48, b = 4, c = 3 > $\log_b a = 2.7924\ldots$
- $T(n) = \Theta(n^3)$.

主定理不适用

主定理中的空白

- 子问题的数量不是一个常数。

$$T(n) = nT(n/2) + n^2$$

- 子问题个数小于1。

$$T(n) = \frac{1}{2}T(n/2) + n^2$$

- 对子问题的划分和组合工作不是$\Theta(n^c)$。

$$T(n) = 2T(n/2) + \boxed{nlogn}$$

分治策略 II：问题 1

考虑下面的递归式，哪种情况属于主定理?

$$T(n) = \begin{cases} 0 & If\ n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & If\ n > 1 \end{cases}$$

A. Case 3: $T(n) = \Theta(n)$.

B. Case 2: $T(n) = \Theta(n\log n)$.

C. Case1: $T(n) = \Theta(n^{\log_3 2}) = O(n^{1.585})$.

D. 主定理不适用

分治策略 II：问题 2

考虑下面的递归式，哪种情况属于主定理?

$$T(n) = \begin{cases} \Theta(1) & If\ n \le 1 \\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \dfrac{11}{5} n & If\ n > 1 \end{cases}$$

A. Case 1: $T(n) = \Theta(n)$.

B. Case 2: $T(n) = \Theta(n\log n)$.

C. Case 3: $T(n) = \Theta(n)$.

D. 主定理不适用

Akra–Bazzi 定理

定理：[Akra–Bazzi 1998] 给定常数$a_i > 0$，$0 < b_i < 1$，

函数$|h_i(n)| = O(n/\log^2 n)$，且$g(n) = O(n^c)$。如果$T(n)$满足递归式：

$$T(n) = \sum_{i=1}^{k} a_i\, T\big(b_i n + h_i(n)\big) + g(n)$$

规模为$b_i n$的a_i个子问题　　处理上下取整的小扰动

则，$T(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right)$，其中$p$满足$\sum_{i=1}^{k} a_i\, {b_i}^p = 1$。

Ex. $T(n) = T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5} n$，其中$T(0) = 0$，$T(1) = 0$。

- $a_1 = 1$, $b_1 = 1/5$, $a_2 = 1$, $b_2 = 7/10 \Rightarrow p = 0.83978 \ldots < 1$
- $h_1(n) = \left\lfloor \frac{n}{5} \right\rfloor - \frac{n}{5}$, $h_2(n) = \frac{3}{10} n - 3\left\lfloor \frac{n}{10} \right\rfloor$
- $g(n) = 11/5n \Rightarrow T(n) = \Theta(n)$

课程提要

- 主定理（Master Theorem）
- 整数乘法（integer multiplication）
- 矩阵乘法（matrix multiplication）

整数加减法

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$a - b$。

Grade-school 算法：$\Theta(n)$位运算 ← "比特复杂性"

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Remark: Grade-school加法和减法是最优的

整数乘法

乘法：给定两个n位整数a和b，计算$a \times b$。

Grade-school 算法 (long multiplication)：$\Theta(n^2)$位运算。

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

推测 [Kolmogorov 1956]: Grade-school algorithms是最优的

定理 [Karatsuba 1960] :推测是错误的

分治乘法

将两个n位整数x和y相乘:

- 把x和y分成低阶和高阶位。
- 递归地乘以四个½n位整数。
- 添加和移动得到结果。

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

use bit shifting
to compute 4 terms

$$xy = (2^m a + b)(2^m c + d) = 2^{2m} \underbrace{ac}_{1} + 2^m (\underbrace{bc}_{2} + \underbrace{ad}_{3}) + \underbrace{bd}_{4}$$

Ex. $x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b} \quad y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$

分治乘法

MULTIPLY(x, y, n)

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

  $m \leftarrow \lceil n / 2 \rceil$.

  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$. $\Theta(n)$

  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

  $e \leftarrow$ MULTIPLY(a, c, m).

  $f \leftarrow$ MULTIPLY(b, d, m). $4\,T(\lceil n / 2 \rceil)$

  $g \leftarrow$ MULTIPLY(b, c, m).

  $h \leftarrow$ MULTIPLY(a, d, m).

  RETURN $2^{2m}\, e + 2^m\, (g + h) + f$. $\Theta(n)$

分治策略 II：问题 3

用分治乘法算法乘两个n位整数需要多少位运算?

$$T(n) = \begin{cases} 0 & If\ n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & If\ n > 1 \end{cases}$$

$A. T(n) = \Theta\left(n^{1/2}\right).$

$B. T(n) = \Theta(n\log n).$

$C. T(n) = \Theta(n^{\log_3 2}) = O(n^{1.585}).$

$D. T(n) = \Theta(n^2).$

Karatsuba 技巧

将两个n位整数x和y相乘:

- 把x和y分成低阶和高阶位。
- 要计算中间项$bc + ad$，使用恒等式:

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

- 递归地乘以三个½n位整数。

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

$$x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b}$$

$$y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$$

middle term

$$x\,y = (2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

1 1 3 2 3

Karatsuba 乘法

$\text{KARATSUBA-MULTIPLY}(x, y, n)$

$\text{IF } (n = 1)$

$\quad \text{RETURN } x \times y.$

ELSE

$\quad m \leftarrow \lceil n / 2 \rceil.$

$\quad a \leftarrow \lfloor x / 2^m \rfloor; \quad b \leftarrow x \bmod 2^m.$ $\qquad \leftarrow \Theta(n)$

$\quad c \leftarrow \lfloor y / 2^m \rfloor; \quad d \leftarrow y \bmod 2^m.$

$\quad e \leftarrow \text{KARATSUBA-MULTIPLY}(a, c, m).$

$\quad f \leftarrow \text{KARATSUBA-MULTIPLY}(b, d, m).$ $\qquad \leftarrow 3\,T(\lceil n / 2 \rceil)$

$\quad g \leftarrow \text{KARATSUBA-MULTIPLY}(|a - b|, |c - d|, m).$

$\quad$ Flip sign of g if needed.

$\quad \text{RETURN } 2^{2m} e + 2^m (e + f - g) + f.$ $\qquad \leftarrow \Theta(n)$

Karatsuba 分析

命题： Karatsuba算法需要$O(n^{1.585})$位运算来计算两个n位整数相乘。

证明： 将主定理的情形3应用于递归式:

$$T(n) = \begin{cases} \Theta(1) & If\ n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & If\ n > 1 \end{cases}$$

$$\Rightarrow \quad T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

实践：

- 使用32或64进位(而不是2进位)。
- 比grade-school算法快约320-640位。

整数乘法渐近复杂度的历史

year	algorithm	bit operations
12xx	grade school	$O(n^2)$
1962	Karatsuba-Ofman	$O(n^{1.585})$
1963	Toom-3, Toom-4	$O(n^{1.465})$, $O(n^{1.404})$
1966	Toom-Cook	$O(n^{1+\varepsilon})$
1971	Schönhage-Strassen	$O(n \log n \cdot \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
2019	Harvey-van der Hoeven	$O(n \log n)$
	???	$O(n)$

两个n位整数相乘的位操作数

注：GNU多精度库使用依赖于n的前五种算法中的一种

GMP
«Arithmetic without limitations»

课程提要

- 主定理（Master Theorem）
- 整数乘法（integer multiplication）
- 矩阵乘法（matrix multiplication）

点积

点积：给定两个长度为n的向量a和b，计算$c = a \cdot b$。

Grade-school：$\Theta(n)$ 算数运算。

$a \cdot b = \sum_{i=1}^{n} a_i b_i$

$$a = [\ .70 \quad .20 \quad .10\ ]$$
$$b = [\ .30 \quad .40 \quad .30\ ]$$
$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

注：“Grade-school”点积是渐进最优的。

矩阵乘法

矩阵乘法：给定两个$n \times n$的矩阵A和B，计算$\underbrace{C = A}_{c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}} \cdot B$。

Grade-school：$\Theta(n^3)$ 算数运算。

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q：“Grade-school”矩阵乘法算法是渐进最优的吗?

分块矩阵乘法

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

C_{11} A_{11} A_{12} B_{11} B_{21}

$$C_{11} = A_{11} + B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

分块矩阵乘法：热身

将两个$n \times n$的矩阵A和B相乘：

- 分解: 将A和B分成$\frac{1}{2}n \times \frac{1}{2}n$块。
- 解决: 递归地乘8对$\frac{1}{2}n \times \frac{1}{2}n$矩阵。
- 组合: 使用4个矩阵加法添加适当的乘积。

n-by-n matrices

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

8 matrix multiplications (of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions (of ½n-by-½n matrices)

运行时间：应用主定理的情形3

$$T(n) = 8T(n/2) + \Theta(n^2) \quad \Rightarrow T(n) = \Theta(n^3)$$

递归调用　　增加，形成矩阵

Strassen's 技巧

关键思想：可以通过7个½n × ½n矩阵乘法(加上11个加法和7个减法)计算矩阵的乘积。

½n-by-½n matrices

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 matrix multiplications
(of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

证明：$C_{12} = P_1 + P_2 = A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} = A_{11} \times B_{12} + A_{12} \times B_{22}$

Strassen's 算法

assume n is a power of 2

STRASSEN(n, A, B)

IF ($n = 1$) RETURN $A \times B$.

Partition A and B into ½n-by-½n blocks.

$P_1 \leftarrow$ STRASSEN($n / 2, A_{11}, (B_{12} - B_{22})$).

$P_2 \leftarrow$ STRASSEN($n / 2, (A_{11} + A_{12}), B_{22}$).

$P_3 \leftarrow$ STRASSEN($n / 2, (A_{21} + A_{22}), B_{11}$).

$P_4 \leftarrow$ STRASSEN($n / 2, A_{22}, (B_{21} - B_{11})$).

$P_5 \leftarrow$ STRASSEN($n / 2, (A_{11} + A_{22}), (B_{11} + B_{22})$).

$P_6 \leftarrow$ STRASSEN($n / 2, (A_{12} - A_{22}), (B_{21} + B_{22})$).

$P_7 \leftarrow$ STRASSEN($n / 2, (A_{11} - A_{21}), (B_{11} + B_{12})$).

$7\,T(n / 2) + \Theta(n^2)$

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

$\Theta(n^2)$

RETURN C.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Strassen's 算法分析

定理: Strassen算法计算两个$n \times n$矩阵相乘需要$O(n^{2.81})$算术运算。

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

Strassen's 算法分析

定理: Strassen算法计算两个$n \times n$矩阵相乘需要$O(n^{2.81})$算术运算。

证明:

- 当n是2的幂时，应用主定理的情形1:

$$T(n) = \underbrace{7T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- 当n不是2的幂时，用0将矩阵填充为$n' \times n'$的矩阵，其中$n \leq n' < 2n$且n'是2的幂。

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

分治策略 II：问题 4

假设你可以用21次标量乘法计算两个3 × 3矩阵相乘。则两个$n \times n$矩阵相乘的速度有多快?

$A.\Theta\left(n^{\log_3 21}\right)$

$B.\Theta\left(n^{\log_2 21}\right)$

$C.\Theta\left(n^{\log_9 21}\right)$

$D.\Theta(n^2)$

矩阵乘法运算复杂度的历史

year	algorithm	arithmetic operations
1858	"grade school"	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith-Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith-Winograd	$O(n^{2.3755})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.372873})$
2014	Le Gall	$O(n^{2.372864})$
	???	$O(n^{2+\varepsilon})$

galactic algorithms

两个$n \times n$矩阵相乘的算术运算数