PKU-04833060
算法设计与分析『实验班』

第3讲 分治策略 (2/2)

罗国杰

gluo@pku.edu.cn

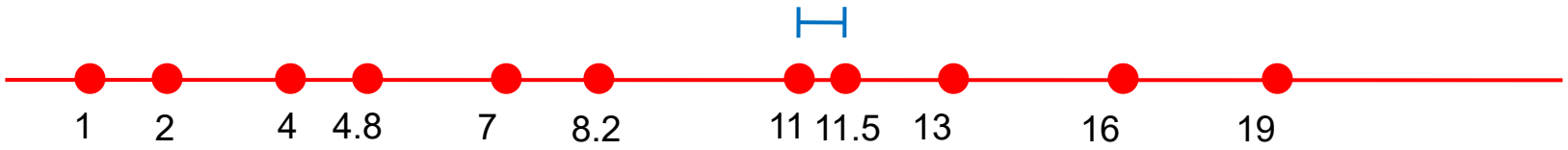
2024年春季学期

改进分治算法的途径

- 递推方程 $T(n) = \Sigma\ a\ T(n/b) + f(n)$
  - 途径1：利用预处理减少递归内部的计算量
    - 例子：平面最邻近点对
    - $T(n) = \Sigma\ a\ T(n/b) + f(n)$
  - 途径2：通过代数变换减少子问题个数
    - 例子：大数乘法和大矩阵乘法
    - $T(n) = \Sigma\ a\ T(n/b) + f(n)$
  - 途径3：减少子问题的总代价
    - 例子：确定性的选择算法
    - $T(n) = \Sigma\ a\ T(n/b) + f(n)$

最近点对（一维）

- 给定直线上 n 个点， 找到距离最近的两个点
  - 例：11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1



- 形状：最近点对在一维序中是相邻的
- 排序、扫描，得到最近点对
- 时间：O(n log n) 排序 + O(n) 扫描

- 关键：利用几何和顺序关系，避免计算所有点对的距离

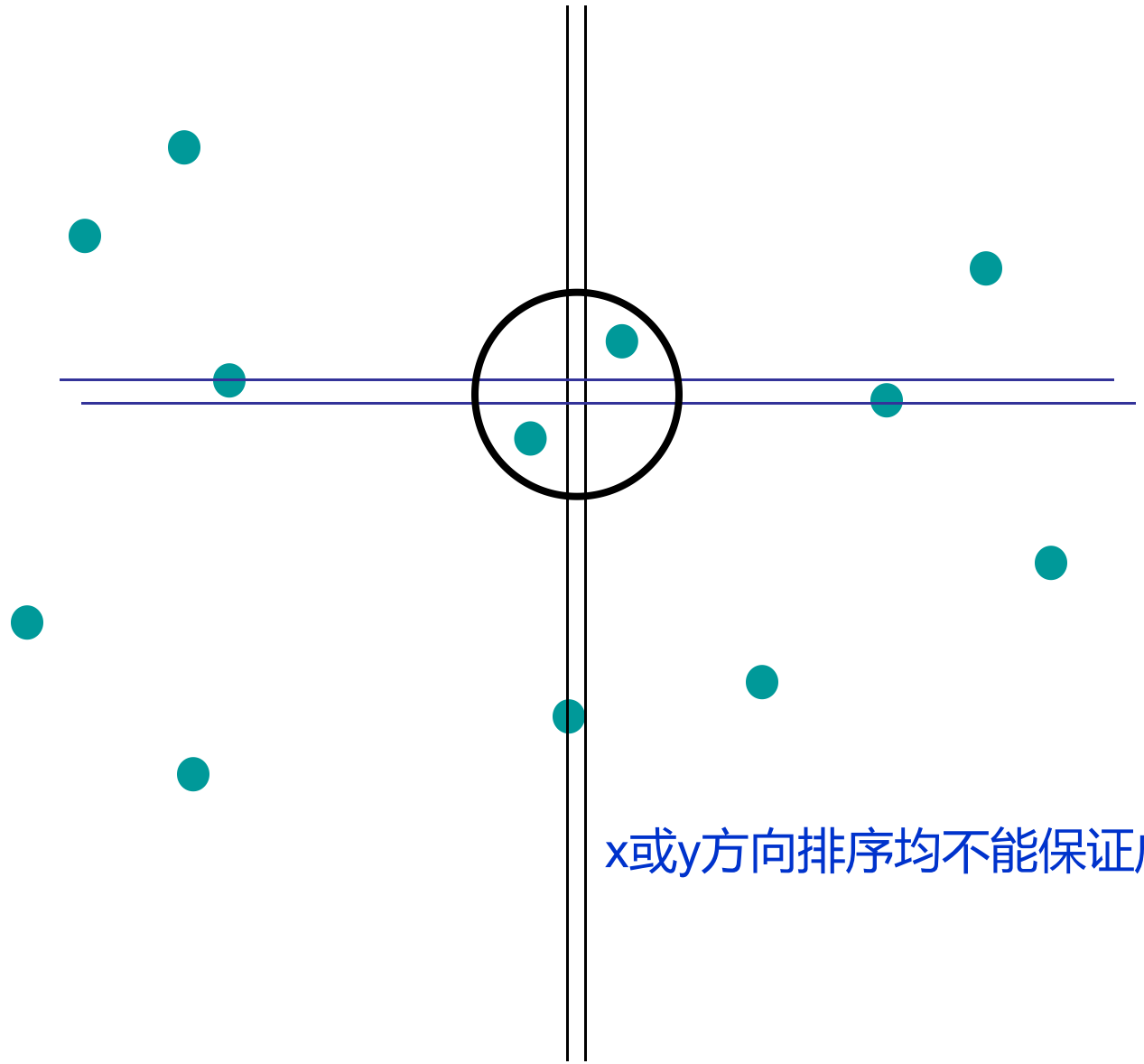
最近点对（二维）

- 给定平面上 n 个点， 找到欧几里得距离最近的两个点
  - 假设：任意两点的 x 坐标均不相同

- 基础的几何原语
  - 1970s，M. I. Shamos 和 D. Hoey 考虑二维最近点对问题，高效的算法成为 计算几何 的基础
  - 图形学、计算机视觉、地理信息系统、分子建模、航空流量控制
  - 最近邻、欧氏最小生成树、Voronoi 图等问题的特例

- 暴力：$\Theta(n^2)$ 的时间检查所有点对的距离

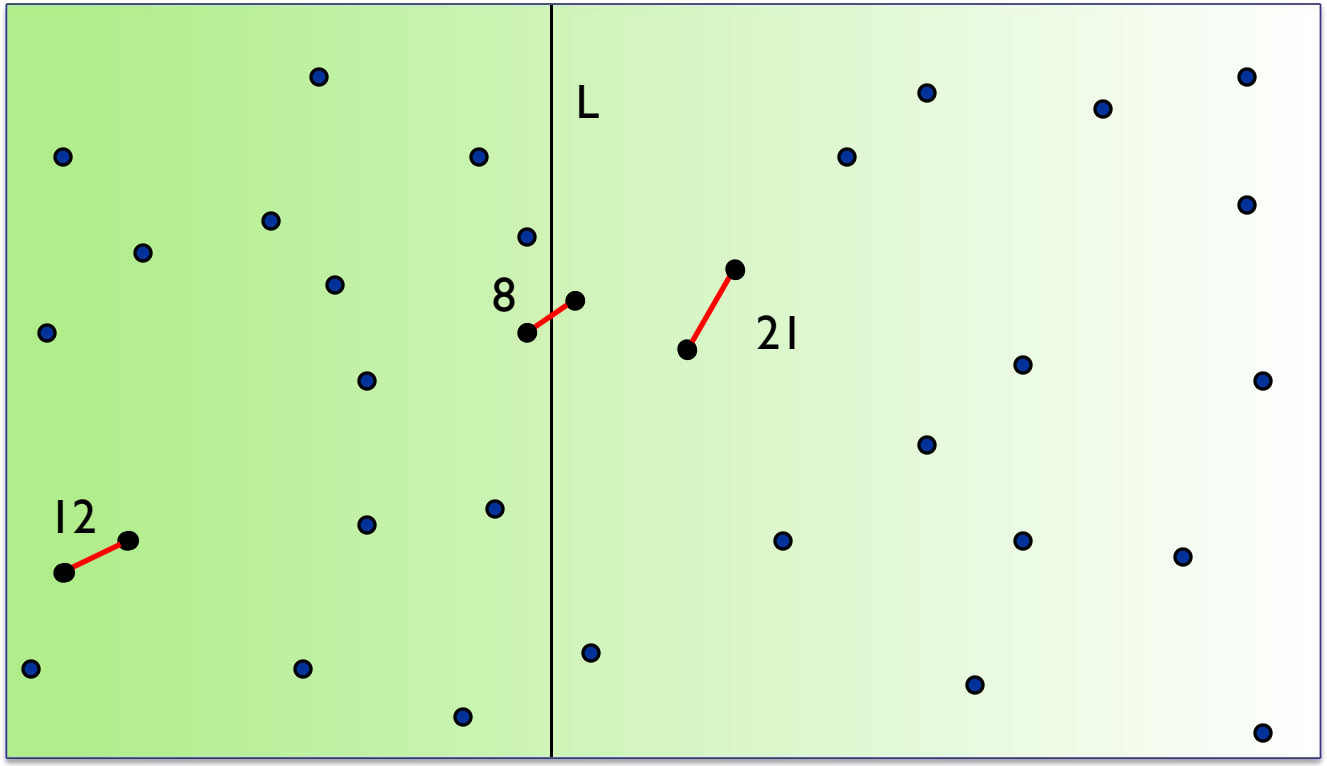
最近点对（二维）



x或y方向排序均不能保证成功求解

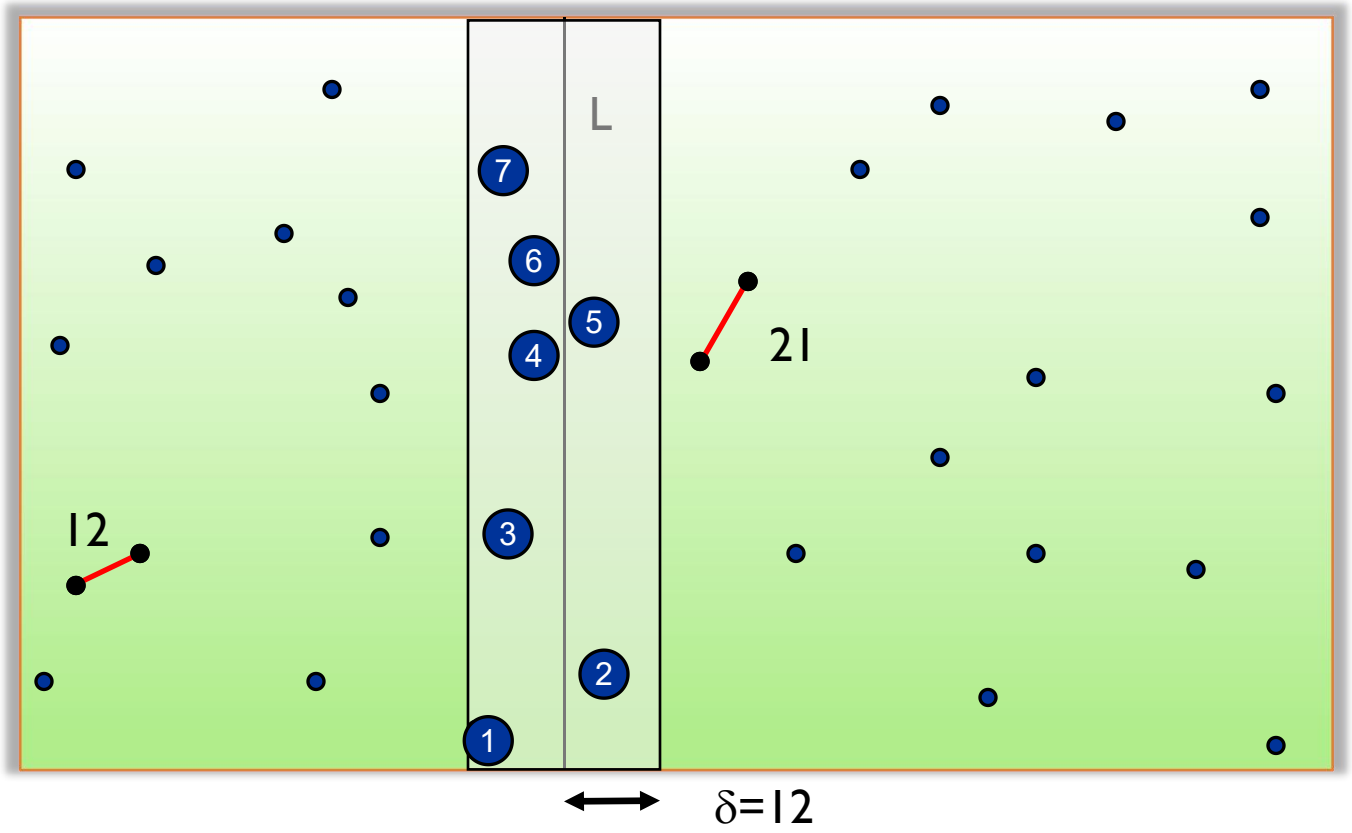
分治策略

- Divide：用竖直线 L 将点集左右等分
- Conquer：左右分别递归查找最近点对
- Combine：求得总体问题的最近点对 ← How?



关键性质

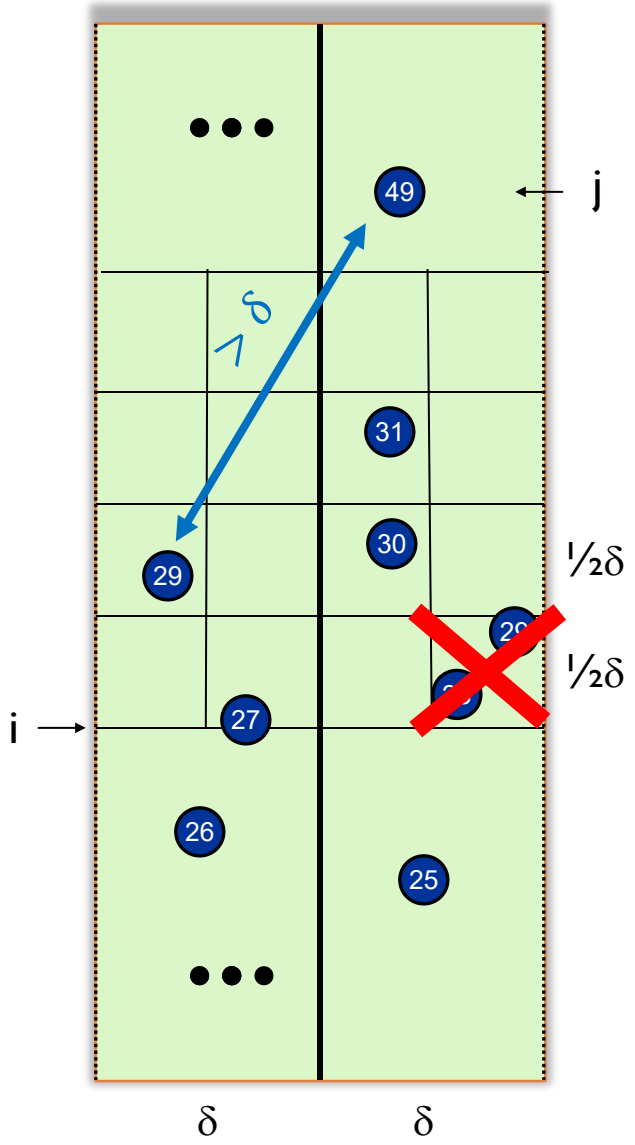
- 假设 δ 是左右集合点对的最小距离。例如：δ=min(12,21)=12
- 关键性质：考虑距离 L 不超过 δ 的点就足够
- 几乎是一维问题：可根据 y 坐标对宽度2δ窄带内的点排序



几乎是一维最近点对

- 沿 L 的宽度 2δ 窄条，左右各画两个宽度 δ/2 的正方形
- 性质：每个 δ/2 × δ/2 正方形内最多只有 1 个点
- 证明：$\sqrt{\left(\frac{\delta}{2}\right)^2+\left(\frac{\delta}{2}\right)^2}=\delta\sqrt{\frac{1}{2}}\approx 0.7\delta<\delta$

- 将宽度 2δ 窄条内的点按 y 坐标排序
- 性质：若序号 |i - j| > 11，则 i 点和 j 点距离 > δ
- 证明：只有 11 个正方形距离 i 点不超过 δ



最近点对（二维）

```
Closest-Pair(p_1, p_2, ..., p_n) {
   if(n ≤ 2) return |p_1 − p_2|

   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ_1 = Closest-Pair(left half)
   δ_2 = Closest-Pair(right half)
   δ   = min(δ_1, δ_2)

   Delete all points further than δ from separation line L

   Sort remaining points p[1]…p[m] by y-coordinate.

   for i = 1, 2, ..., m
      for k = 1, 2, ..., 11
        if i + k ≤ m
            δ = min(δ, distance(p[i], p[i+k]));

   return δ.
}
```

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n \log n) & \text{o.w.} \end{cases}$$

$$\Rightarrow T(n) = O(n \log^2 n)$$

最近点对（二维）改进

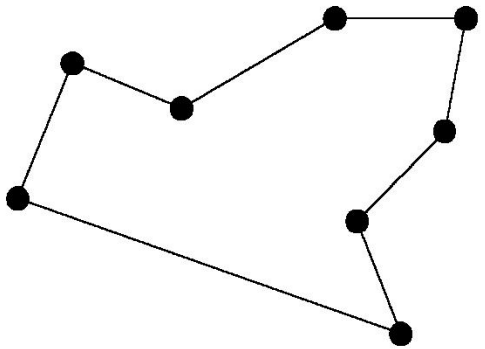
```
Closest-Pair(p_1, p_2, ..., p_n) {
    if(n ≤ 2) return |p_1 − p_2|

    Compute separation line L such that half the points
    are on one side and half on the other side.

   (δ_1, p_1) = Closest-Pair(left half)
   (δ_2, p_2) = Closest-Pair(right half)
    δ         = min(δ_1, δ_2)
    p_sorted  = merge(p_1, p_2)   (merge sort it by y-coordinate)

    Let q be points (ordered as p_sorted) that is δ from line L.

    for i = 1, 2, ..., m
        for k = 1, 2, ..., 11
          if i + k ≤ m
              δ = min(δ, distance(q[i], q[i+k]));

    return δ and p_sorted.
}
```

$$T(n) \le \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & \text{o.w.} \end{cases}$$

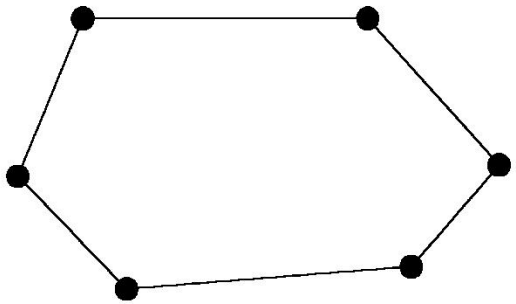
$$\Rightarrow T(n) = O(n \log n)$$

平面凸包问题

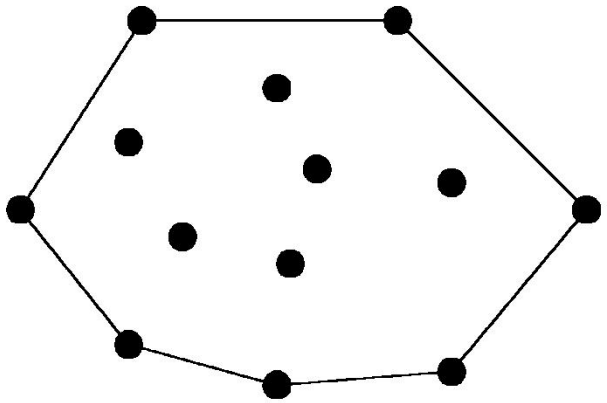
凹多边形



凸多边形

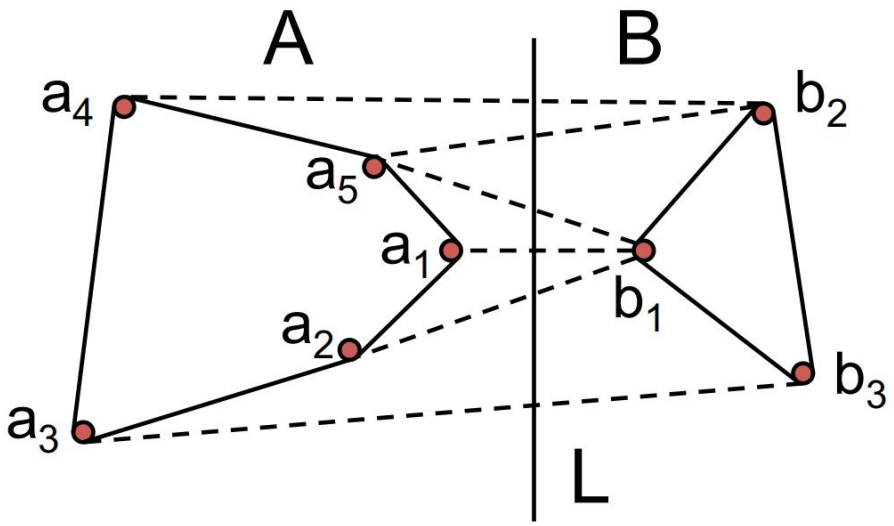


- 平面点集的凸包，是包含此点集的最小的多边形



平面凸包问题

- 分治策略



  - 暴力合并 $\Theta(n^2)$
  - 暴力分治 $T(n) = 2\,T(n/2) + \Theta(n^2) = \Theta(n^2)$

- 利用性质：(ai, bj) 是上切线，当且仅当它与 L 交点的 y 坐标 y(i,j) 最大
- O(n) 合并算法
  - 求上切线：A 最右侧 a1 起逆时针移动、或 B 最左侧 b1 起顺时针移动，找使 y(i,j) 最大的 (ai, bj)
  - 求下切线类似
- $\Theta(n \log n)$ 分治
  - $T(n) = 2\,T(n/2) + \Theta(n) = \Theta(n \log n)$

改进分治算法的途径

- 通过代数变换 减少子问题个数
  - T(n) = **a** T(n/b) + f(n)
- 例：大整数乘法、矩阵乘法

整数算术

- 加法: 给定两个n位整数 a 和 b
- 计算 a+b。

$O(n)$ 个位运算

Add

	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

- 乘法：给定两个n位整数 a 和 b
- 计算 a×b。竖式乘法：

$O(n^2)$ 个位运算

Multiply

					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							
0	0	0	0	0	0	0	0								
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

整数乘法：分治策略

- 给定两个n位整数x, y
- 记 $x = 2^{n/2}x_1 + x_0$ 和 $y = 2^{n/2}y_1 + y_0$
  - 其中 x_0, x_1, y_0, y_1 均为 $n/2$ 位整数

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
\end{aligned}
$$

只需三部分的值
$x_1y_1, x_0y_0, x_1y_0 + x_0y_1$
能否只用三个乘法?

- 因此，$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^2)$

整数乘法：将四次乘法降为三次

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0)$$
$$= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0$$

$$\alpha = x_1 + x_0$$
$$\beta = y_1 + y_0$$
$$\alpha\beta = (x_1 + x_0)(y_1 + y_0)$$
$$= x_1 y_1 + (x_1 y_0 + x_0 y_1) + x_0 y_0$$
$$(x_1 y_0 + x_0 y_1) = \alpha\beta - x_1 y_1 - x_0 y_0$$

整数乘法：将四次乘法降为三次

- 定理 [Karatsuba-Ofman, 1962]：两个n位整数相乘的复杂度是 $O(n^{1.585\ldots})$ 次位运算

$$x = 2^{n/2} \cdot x_1 + x_0 \Rightarrow \alpha = x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0 \Rightarrow \beta = y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= \underbrace{2^n \cdot x_1 y_1}_{A} + \underbrace{2^{n/2} \cdot (x_1 y_0 + x_0 y_1)}_{\alpha\beta - A - B} + \underbrace{x_0 y_0}_{B}$$

- 两个n位整数相乘
  - 求两次 $n/2$ 位整数加法
  - 求三次 $n/2$ 位整数乘法
  - $n/2$ 整数的加法/减法/移位

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O\left(n^{\log_2 3}\right) = O(n^{1.585\ldots})$$

整数乘法：汇总

- 练习：将 Karatsuba 算法改为5个$n/3$大小子问题的分治
  - $\Theta(n^{1.46\ldots})$ 时间算法

Date	Authors	Time complexity
<3000 BC	Unknown	$O(n^2)$
1962	Karatsuba	$O(n^{\log 3/\log 2})$
1963	Toom	$O(n\,2^{5\sqrt{\log n/\log 2}})$
1966	Schönhage	$O(n\,2^{\sqrt{2\log n/\log 2}}\,(\log n)^{3/2})$
1969	Knuth	$O(n\,2^{\sqrt{2\log n/\log 2}}\log n)$
1971	Schönhage–Strassen	$O(n\log n\log\log n)$
2007	Fürer	$O\left(n\log n\,2^{O(\log^* n)}\right)$
2014	Harvey-Hoeven-Lecerf	$O(n\log n\,8^{\log^* n})$
2019	Harvey-Hoeven	$O(n\log n)$

https://courses.cs.washington.edu/courses/cse421/22wi/lecture/lecture-12.pptx

矩阵乘法

例　A,B 为两个n 阶矩阵，$n=2^k$, 计算 $C=AB$.

传统算法　$W(n)=O(n^3)$

分治法　将矩阵分块，得

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

其中

$$C_{11}=A_{11}B_{11}+A_{12}B_{21} \qquad C_{12}=A_{11}B_{12}+A_{12}B_{22}$$

$$C_{21}=A_{21}B_{11}+A_{22}B_{21} \qquad C_{22}=A_{21}B_{12}+A_{22}B_{22}$$

递推方程　$W(n)=8\,W(n/2)+cn^2$

$$W(1)=1$$

解　　$W(n)=O(n^3)$.

Strassen 矩阵乘法

变换方法：

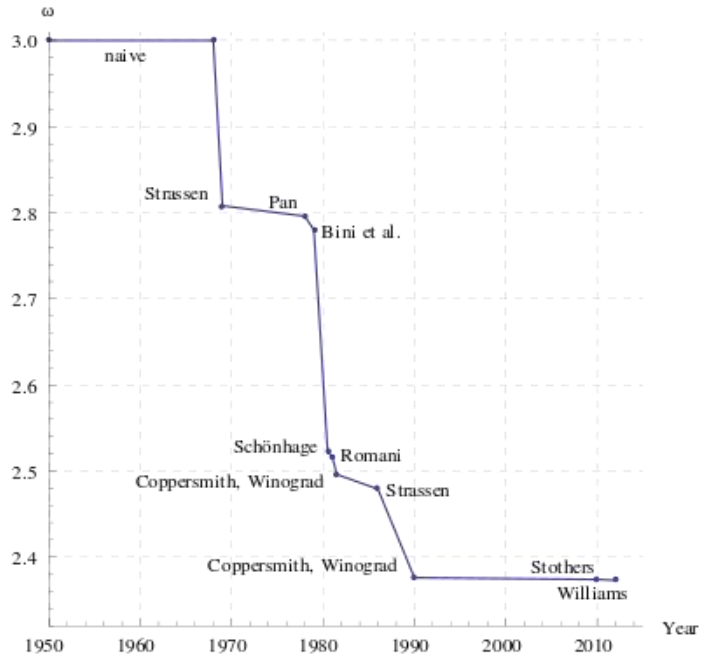
$$M_1 = A_{11}(B_{12} - B_{22})$$
$$M_2 = (A_{11} + A_{12})B_{22}$$
$$M_3 = (A_{21} + A_{22})B_{11}$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$C_{11} = M_5 + M_4 - M_2 + M_6$$
$$C_{12} = M_1 + M_2$$
$$C_{21} = M_3 + M_4$$
$$C_{22} = M_5 + M_1 - M_3 - M_7$$

时间复杂度：

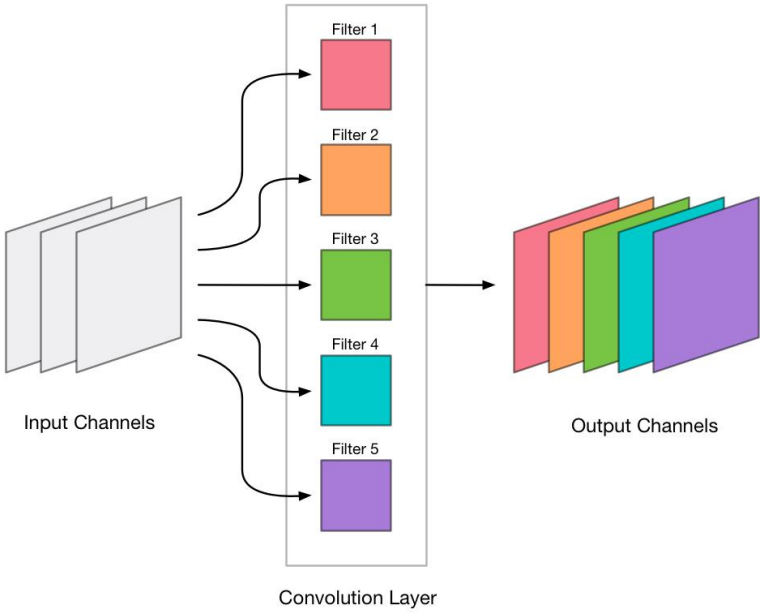
$$W(n) = 7W(\frac{n}{2}) + 18(\frac{n}{2})^2$$
$$W(1) = 1$$

$$W(n) = O(n^{\log_2 7}) = O(n^{2.8075})$$



Strassen方法应用于卷积 [ICANN14]

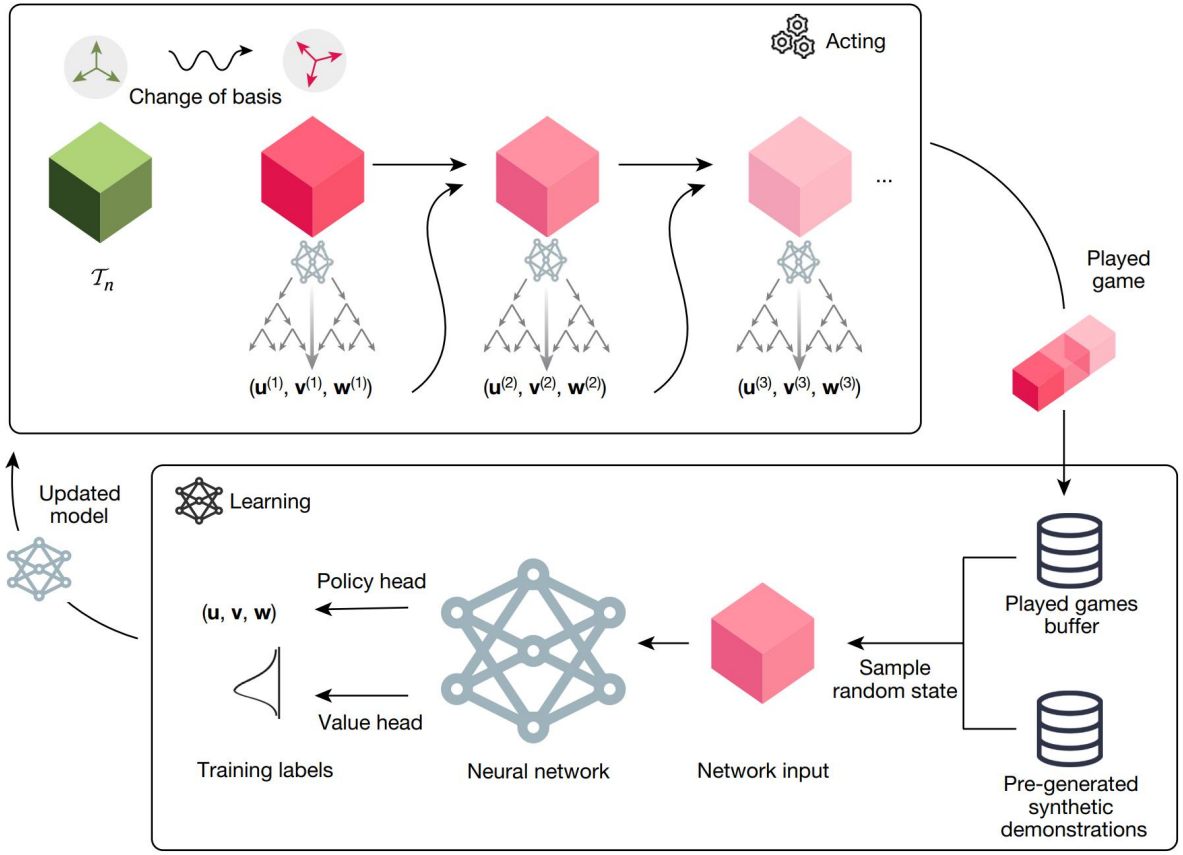
- 由 Q 组输入卷积得到 R 组输出
  - 输出 y_i 依赖于输入 $\{x_k\}$ 以及卷积核 $\{W_{i,k}\}$



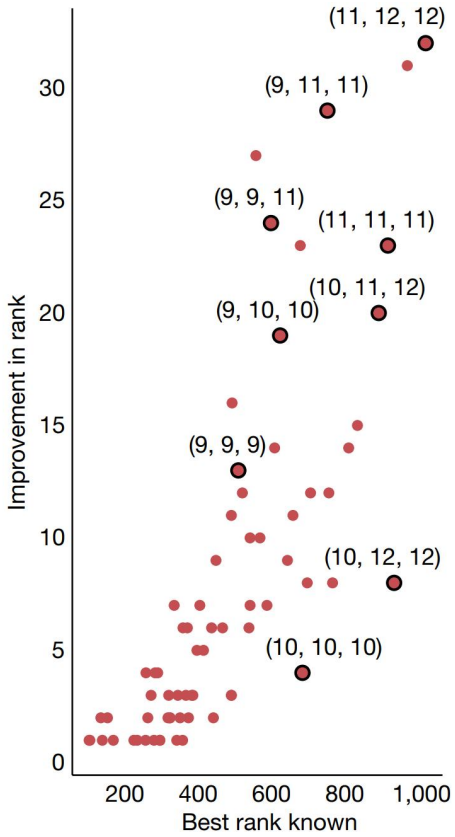
$$
\begin{aligned}
y_1 &= w_{11} * x_1 + w_{12} * x_2 + \cdots w_{1Q} * x_Q \\
y_2 &= w_{21} * x_1 + w_{22} * x_2 + \cdots w_{2Q} * x_Q \\
y_3 &= w_{31} * x_1 + w_{32} * x_2 + \cdots w_{3Q} * x_Q \\
&\cdots \\
y_R &= w_{R1} * x_1 + w_{R2} * x_2 + \cdots w_{RQ} * x_Q
\end{aligned}
$$

$\vec{y} = W \times \vec{x}$ （概念上“重载”×算符，来推导分治策略）

AlphaTensor：搜索具体实例的较优解



Size (n, m, p)	Best method known	Best rank known	AlphaTensor rank Modular	AlphaTensor rank Standard
(2, 2, 2)	(Strassen, 1969)[2]	7	7	7
(3, 3, 3)	(Laderman, 1976)[15]	23	23	23
(4, 4, 4)	(Strassen, 1969)[2] $(2,2,2) \otimes (2,2,2)$	49	47	49
(5, 5, 5)	(3, 5, 5) + (2, 5, 5)	98	96	98
(2, 2, 3)	(2, 2, 2) + (2, 2, 1)	11	11	11
(2, 2, 4)	(2, 2, 2) + (2, 2, 2)	14	14	14
(2, 2, 5)	(2, 2, 2) + (2, 2, 3)	18	18	18
(2, 3, 3)	(Hopcroft and Kerr, 1971)[16]	15	15	15
(2, 3, 4)	(Hopcroft and Kerr, 1971)[16]	20	20	20
(2, 3, 5)	(Hopcroft and Kerr, 1971)[16]	25	25	25
(2, 4, 4)	(Hopcroft and Kerr, 1971)[16]	26	26	26
(2, 4, 5)	(Hopcroft and Kerr, 1971)[16]	33	33	33
(2, 5, 5)	(Hopcroft and Kerr, 1971)[16]	40	40	40
(3, 3, 4)	(Smirnov, 2013)[18]	29	29	29
(3, 3, 5)	(Smirnov, 2013)[18]	36	36	36
(3, 4, 4)	(Smirnov, 2013)[18]	38	38	38
(3, 4, 5)	(Smirnov, 2013)[18]	48	47	47
(3, 5, 5)	(Sedoglavic and Smirnov, 2021)[19]	58	58	58
(4, 4, 5)	(4, 4, 2) + (4, 4, 3)	64	63	63
(4, 5, 5)	(2, 5, 5) $\otimes$ (2, 1, 1)	80	76	76



Fawzi, A. et al., (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 610(7930), 47–53.

分治策略应用：卷积

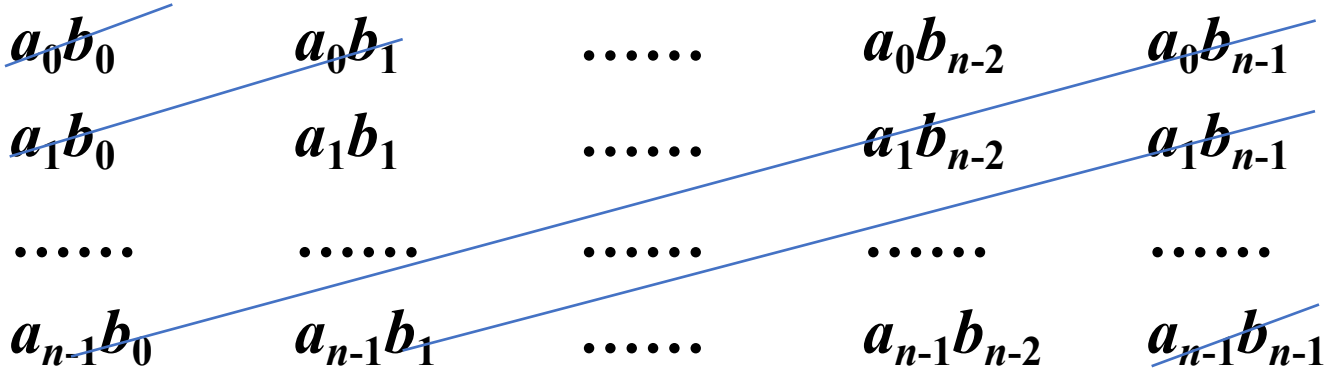
- 向量计算：

给定向量 $a = (a_0, a_1, \ldots, a_{n-1})$ 和 $b = (b_0, b_1, \ldots, b_{n-1})$

向量和　$a + b = (a_0+b_0, a_1+b_1, \ldots, a_{n-1}+b_{n-1})$

内积　　$a \cdot b = a_0b_0 + a_1b_1 + \ldots + a_{n-1}b_{n-1}$

卷积　　$a * b = (c_0, c_1, \ldots, c_{2n-2})$，其中 $c_k = \sum_{\substack{i+j=k \\ i,j<n}} a_i b_j, \quad k = 0,1,\ldots,2n-2$



卷积计算的例子

$a=(a_0,a_1,\ldots,a_8)$, $k=2$, $w=(w_{-2},w_{-1},w_0,w_1,w_2)=(b_0,b_1,b_2,b_3,b_4)$

$a_i'=a_{i-2}b_4+a_{i-1}b_3+a_ib_2+a_{i+1}b_1+a_{i+2}b_0$，下标之和为 $i+k$

a_0b_0	a_0b_1	a_0b_2	a_0b_3	a_0b_4	a_2'
a_1b_0	a_1b_1	a_1b_2	a_1b_3	a_1b_4	a_3'
a_2b_0	a_2b_1	a_2b_2	a_2b_3	a_2b_4	a_4'
a_3b_0	a_3b_1	a_3b_2	a_3b_3	a_3b_4	a_5'
a_4b_0	a_4b_1	a_4b_2	a_4b_3	a_4b_4	a_6'
a_5b_0	a_5b_1	a_5b_2	a_5b_3	a_5b_4	
a_6b_0	a_6b_1	a_6b_2	a_6b_3	a_6b_4	
a_7b_0	a_7b_1	a_7b_2	a_7b_3	a_7b_4	
a_8b_0	a_8b_1	a_8b_2	a_8b_3	a_8b_4	

平滑处理

- 定义向量

$$a = (a_0, a_1, \cdots, a_{m-1})$$
$$b = (b_0, b_1, \cdots, b_{2k}) = (w_{-k}, w_{-k+1}, \cdots, w_k) = w$$

$$a_i' = \sum_{s=-k}^{k} a_{i-s} b_{s+k} = \sum_{s=-k}^{k} a_{i-s} w_s$$

  - 把权向量 b=w 看作2k+1长度的窗口在a上移动计算卷积.

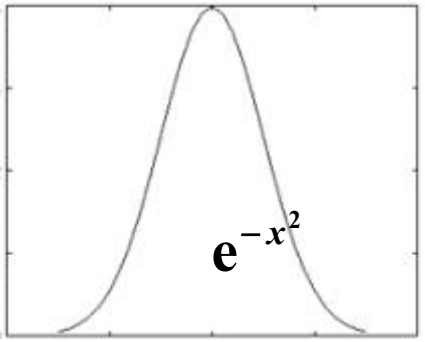
卷积应用：信号处理

给定序列$a=(a_0,a_1,\ldots,a_{m-1})$，表示信号在时刻0,1,…,$m$-1的度量值，由于噪音干扰，需要平滑处理, 即对值 a_i，用其前后k步内的值进行加权平均. 离a_i越近权值越大，处理结果为 a_i'.

高斯滤波的权值函数为

$$w_s = \frac{1}{z}\mathrm{e}^{-s^2}, \quad s = 0, \pm 1, ..., \pm k$$

$$w = (w_{-k}, ..., w_{-1}, w_0, w_1, ..., w_k)$$



其中z用于归一化处理，使得所有的权值之和为1.

处理结果为

$$a_i' = \sum_{s=-k}^{k} a_{i-s} w_s$$

快速卷积的分治算法

- 基本思想
  - N维向量 ⇔ (N-1)次多项式 ~生成函数
  - 向量卷积 ⇔ 多项式乘积
  - (N-1)次多项式 ⇔ 0次项至(N-1)次项的系数 ⇔ N个插值点

  - N维向量 → 多项式乘积 → 插值点标量乘
  - 通过插值点重建目标多项式系数
    - 利用1的多次方根在多项式值的特殊性质减少计算量
    - 构造特殊多项式的插值点求得目标多项式的各项系数

可阅读：Kleinberg, J., & Tardos, E. (2006). Algorithm Design. Pearson Education 的 第5.6节 Convolutions and the Fast Fourier Transform

卷积等价于多项式相乘

多项式乘法：

给定多项式

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$$

计算 $C(x) = A(x)\,B(x)$

$$= a_0 b_0 + (a_0 b_1 + a_1 b_0)\, x + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, x^2$$

$$+ \ldots + a_{m-1} b_{n-1}\, x^{m+n-2}$$

其中 x^k 的系数 $\boldsymbol{c_k = \sum_{\substack{i+j=k \\ i\in\{0,1,\ldots,m-1\} \\ j\in\{0,1,\ldots,n-1\}}} a_i b_j, \quad k = 0,1,\ldots,m+n-2}$

卷积计算

给定向量 $a = (a_0, a_1, \ldots, a_{n-1})$和 $b = (b_0, b_1, \ldots, b_{n-1})$

直接计算：$O(n^2)$

快速算法：卷积计算等价于多项式相乘.

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

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

$$C(x)=A(x)B(x)= a_0b_0 + (a_0b_1+a_1b_0)\,x + (a_0b_2+a_1b_1+a_2b_0)\,x^2$$

$$+ \quad (\ldots + a_{m-1}b_{n-1})\,x^{m+n-2}$$ ， $C(x)$的系数向量就是$a * b$.

设计思想：

1. 选择值$x_1, x_2, \ldots, x_{2n}$，求出$A(x_j)$和$B(x_j)$，j=1,2,…,2n
2. 对每个j，计算$C(x_j)$
3. 利用多项式插值方法，根据$C(x)$在$x=x_1,x_2,\ldots, x_{2n}$的值求出多项式$C(x)$

多项式求值算法

给定多项式：

设 x 为 1 的 $2n$ 次方根，对所有的 x 计算 $A(x)$ 的值.

算法1：对每个 x 做下述运算：

依次计算每个项 $a_i x^i$，对 i 求和得到 $A(x)$，

$T_1(n)=O(n^3)$（每个$A(x_k)$时间$O(n^2)$，共$2n$个$\{x_k\}$；总时间$O(n^3)$）

算法2：$A_1(x)= a_{n-1}$

$A_2(x)= a_{n-2}+ xA_1(x)$

$A_3(x)= a_{n-3}+ xA_2(x)$

…

$A_n(x)= a_0+ xA_{n-1}(x)=A(x)$

$T_2(n) = O(n^2)$（每个$A(x_k)$时间$O(n)$，共$2n$个$\{x_k\}$；总时间$O(n^2)$；未利用$2n$个x的关系）

2n个数的选择：1 的2n次根

$$\omega_j = \mathrm{e}^{\frac{2\pi j}{2n}\mathrm{i}} = \mathrm{e}^{\frac{\pi j}{n}\mathrm{i}} = \cos\frac{\pi j}{n} + \mathrm{i}\sin\frac{\pi j}{n} \qquad j=0,1,\ldots,2n-1$$

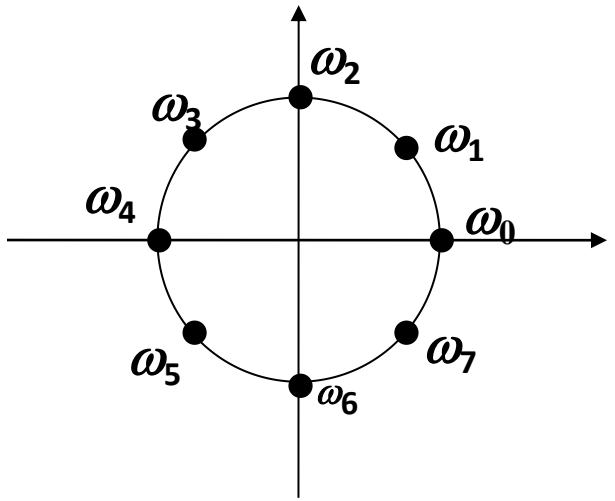
例如n=4, 1的8次方根是：

$$\omega_0 = 1, \qquad \omega_1 = \mathrm{e}^{\frac{\pi}{4}\mathrm{i}} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}\mathrm{i},$$

$$\omega_2 = \mathrm{e}^{\frac{\pi}{2}\mathrm{i}} = \mathrm{i}, \quad \omega_3 = \mathrm{e}^{\frac{3\pi}{4}\mathrm{i}} = -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}\mathrm{i},$$

$$\omega_4 = \mathrm{e}^{\pi\mathrm{i}} = -1, \ \omega_5 = \mathrm{e}^{\frac{5\pi}{4}\mathrm{i}} = -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}\mathrm{i},$$

$$\omega_6 = \mathrm{e}^{\frac{3\pi}{2}\mathrm{i}} = -\mathrm{i}, \ \omega_7 = \mathrm{e}^{\frac{7\pi}{4}\mathrm{i}} = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}\mathrm{i}$$



分治法：多项式求值

原理：

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

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

$A(x)=A_{even}(x^2)+x\,A_{odd}(x^2)$，　x^2 为 1 的 n 次根

算法3 时间复杂度$T(n)$

1. 计算 1 的 $2n$ 次根 $\{\omega_0, \ldots, \omega_{2n-1}\}$
2. 算法3.1计算$A(x)$在$\{\omega_0, \ldots, \omega_{2n-1}\}$的值

算法3.1 时间复杂度$T_1(n)$

（计算$A(x)$在2n个插值点的值）

输入 2n个插值点$\{\omega_0, \omega_1, \ldots, \omega_{2n-1}\}$

$A(x) \sim \{a_0, a_1, \ldots, a_{n-1}\}$

输出 $\{A(\omega_0), A(\omega_1), \ldots, A(\omega_{2n-1})\}$

1. 算法3.1计算$A_{even}(x)$在n个插值点$\{\omega_0, \omega_2, \ldots, \omega_{2n-2}\}$的值
2. 算法3.1计算$A_{odd}(x)$在n个插值点$\{\omega_0, \omega_2, \ldots, \omega_{2n-2}\}$的值
3. 计算 $A(x)=A_{even}(x^2)+x\,A_{odd}(x^2)$ 在2n个插值点 $\{\omega_0, \omega_1, \ldots, \omega_{2n-1}\}$ 的值

（利用 x^2 为 1 的 n 次根的性质，以及步骤1和2的结果）

复杂度分析：

$T(n)=T_1(n)+f(n)$，
$f(n)=O(n)$计算$2n$次根时间

$T_1(n)=2T_1(n/2)+O(n)$,

$T_1(1)=O(1)$

$T(n)=O(n\log n)$

卷积计算的时间分析

步1：求值 $A(\omega_j)$ 和 $B(\omega_j)$，$j=0,1,\ldots,2n-1$. $O(n\log n)$

步2：计算 $C(\omega_j)$，$j=0,1,\ldots,2n-1$. $O(n)$

步3：构造多项式

$$D(x)=C(\omega_0)+C(\omega_1)x+\ldots+C(\omega_{2n-1})x^{2n-1}$$

可以证明： $D(\omega_0)=2nc_0$

$D(\omega_j)=2nc_{2n-j}$，j=1 ,…,$2n-1$

即只要计算出所有的 $D(\omega_j)$，就得到所有的c_{2n-j}，$j=0,1,\ldots,2n-1$. $O(n\log n)$

总的时间为$O(n\log n)$

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents **Quicksort**.

Putting N things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Although it's possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first item on a list that's already sorted!), Quicksort runs on average with $O(N \log N)$ efficiency. Its elegant simplicity has made Quicksort the pos-terchild of computational complexity.

James Cooley

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the **fast Fourier transform**.

Easily the most far-reaching algo-rithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quick- sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

John Tukey

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an **integer relation detection algorithm**.

The problem is an old one: Given a bunch of real numbers, say $x_1, x_2, \ldots, x_n$, are there integers $a_1, a_2, \ldots, a_n$ (not all 0) for which $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$? For $n = 2$, the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of x_1/x_2. If x_1/x_2 is rational, the expansion terminates and, with proper unraveling, gives the "smallest" integers a_1 and a_2. If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, $B_3 = 3.544090$ and $B_4 = 3.564407$, of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 257^{30}.) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the **fast multipole algorithm**.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations—one for each pair of particles. The fast multipole algorithm gets by with $O(N)$ computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

分治策略

内容小结

- 分治算法例子
  - 芯片测试、幂乘计算
  - Karatsuba算法和Strassen算法
- 改进分治算法：$T(n) = a\,T(n/b) + f(n)$
  - 减少子问题个数、降低子问题规模、优化预处理和后处理时间
  - 例子：平面最近点对、分治选择、分治卷积