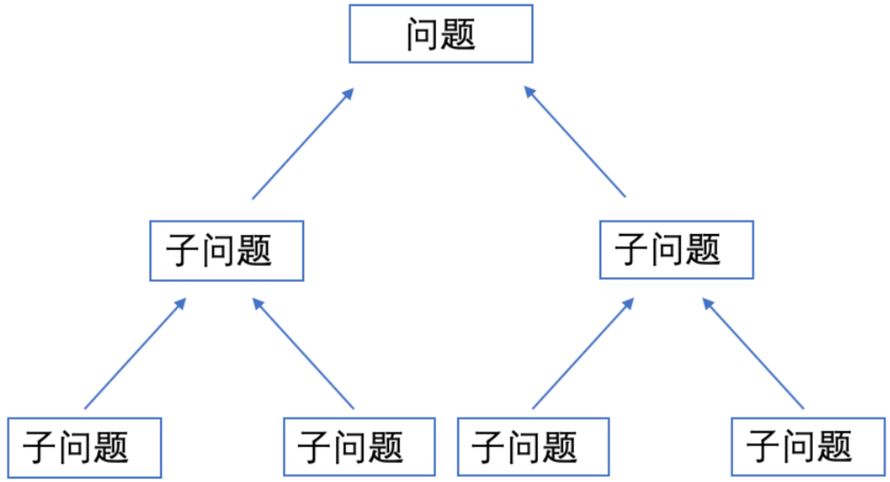
数据分析与算法设计
第5章 分治法
（Divide-and-Conquer）

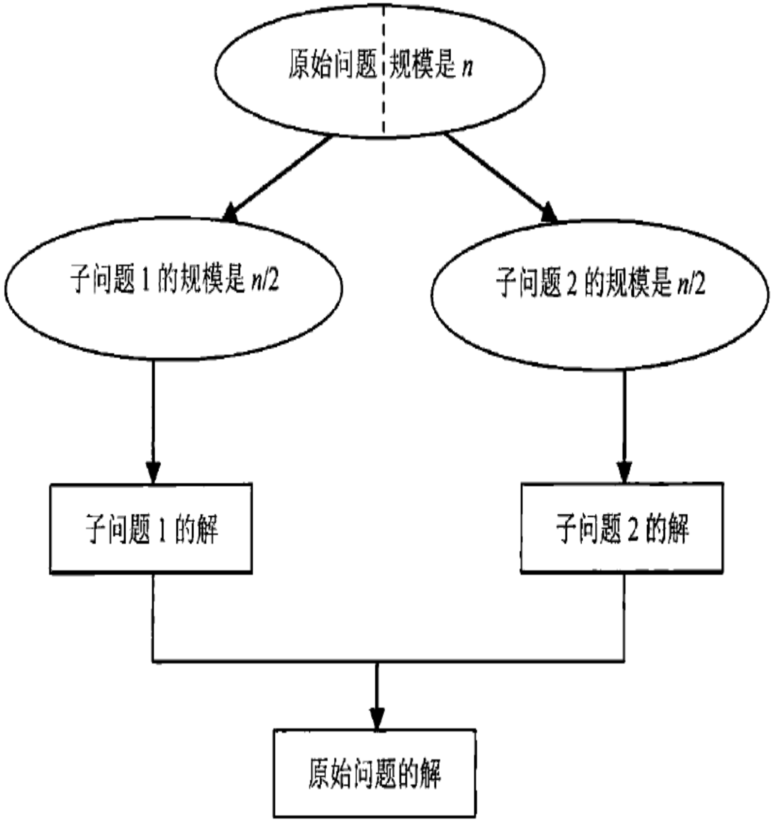
李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

分治法

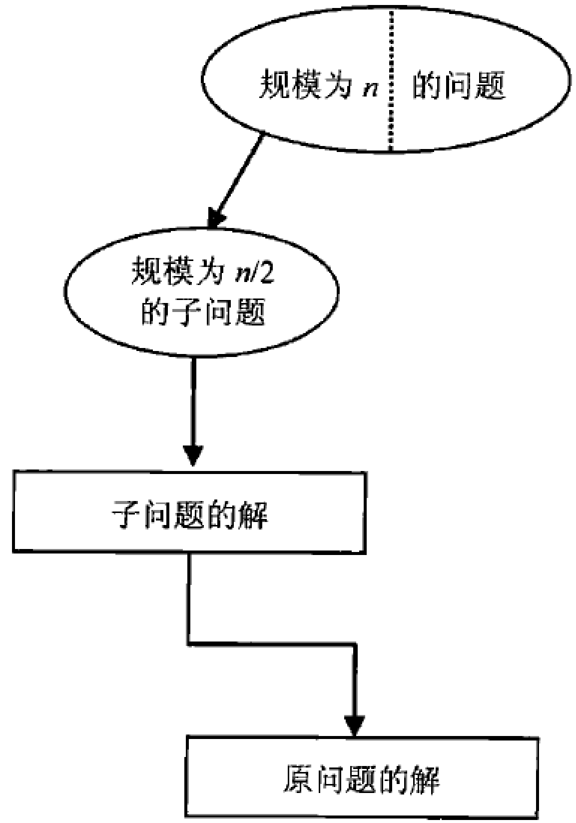
- 基本思想：将问题规模为n的问题分解为b个规模较小的子问题，分别求解每个子问题，再将子问题的解合并成原问题的解
  - 如子问题的规模仍较大时，则反复分解直到问题小到可以求解
  - 子问题的解法一般与原问题相同，自然形成递归形式



分治 vs 减治



分治



减（半）治

分治法：效率通用递推式

$T(n) = aT(n/b) + f(n)$ 其中，$f(n) \in \Theta(n^d)$，$d \geq 0$

主定理:　当 $a < b^d$，$T(n) \in \Theta(n^d)$

当 $a = b^d$，$T(n) \in \Theta(n^d \log n)$

当 $a > b^d$，$T(n) \in \Theta(n^{\log_b a})$

注意：对符号O和Ω，类似的结论也成立

例如: $T(n) = 4T(n/2) + n \Rightarrow T(n) \in ?$

$T(n) = 4T(n/2) + n^2 \Rightarrow T(n) \in ?$

$T(n) = 4T(n/2) + n^3 \Rightarrow T(n) \in ?$

分治法：效率通用递推式

- 证明思路

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

$$\Downarrow \; n = b^k$$

$$\begin{aligned}
T(b^k) &= aT(b^{k-1}) + f(b^k) \\
&= a[aT(b^{k-2}) + f(b^{k-1})] + f(b^k) = a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k) \\
&= a^2[aT(b^{k-3}) + f(b^{k-2})] + af(b^{k-1}) + f(b^k) \\
&= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k) \\
&= \cdots \\
&= a^kT(1) + a^{k-1}f(b^1) + a^{k-2}f(b^2) + \cdots + a^0f(b^k) \\
&= a^k[T(1) + \sum_{j=1}^{k} f(b^j)/a^j].
\end{aligned}$$

$$\Downarrow$$

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j].$$

分治法：效率通用递推式

- 当 $f(n) = n^d$

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} b^{jd}/a^j] = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j]$$

If $a < b^d$, then $b^d/a > 1$, and therefore

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = (b^d/a)\frac{(b^d/a)^{\log_b n} - 1}{(b^d/a) - 1} \in \Theta((b^d/a)^{\log_b n}).$$

Hence, in this case,

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j] \in n^{\log_b a}\Theta((b^d/a)^{\log_b n})$$

$$= \Theta(n^{\log_b a}(b^d/a)^{\log_b n}) = \Theta(a^{\log_b n}(b^d/a)^{\log_b n})$$

$$= \Theta(b^{d \log_b n}) = \Theta(b^{\log_b n^d}) = \Theta(n^d).$$

其它情况，可
采用类似推导

目录

- 合并排序
- 快速排序
- 二叉树遍历
- 大整数乘法和Strassen矩阵乘法
- 最近对问题和凸包问题

合并排序

- 算法思想
  - 将要排序的数组分割成k个子数组(k一般为2),对每个子数组分别进行递归排序后,再将排好的子数组合并为一个元素非递减的数组
- 算法伪代码

算法 Mergesort($A[0..n-1]$)

//递归调用 mergesort 来对数组 $A[0..n-1]$ 排序

//输入：一个可排序数组 $A[0..n-1]$

//输出：非降序排列的数组 $A[0..n-1]$

if $n > 1$

  copy $A[0..\lfloor n/2 \rfloor - 1]$ **to** $B[0..\lfloor n/2 \rfloor - 1]$

  copy $A[\lfloor n/2 \rfloor..n-1]$ **to** $C[0..\lceil n/2 \rceil - 1]$

  Mergesort($B[0..\lfloor n/2 \rfloor - 1]$)

  Mergesort($C[0..\lceil n/2 \rceil - 1]$)

  Merge(B, C, A) //参见下文

合并排序

算法 Merge($B[0..p-1]$, $C[0..q-1]$, $A[0..p+q-1]$)

//将两个有序数组合并为一个有序数组

//输入：两个有序数组 $B[0..p-1]$和 $C[0..q-1]$

//输出：$A[0..p+q-1]$中已经有序存放了 B 和 C 中的元素

$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$

while $i < p$ **and** $j < q$ **do**

  if $B[i] \leqslant C[j]$

    $A[k] \leftarrow B[i]; i \leftarrow i+1$ ——→ 将较小的元素添加到一个输出数组中，被复制数组的元素下标后移

  else $A[k] \leftarrow C[j]; j \leftarrow j+1$

  $k \leftarrow k+1$

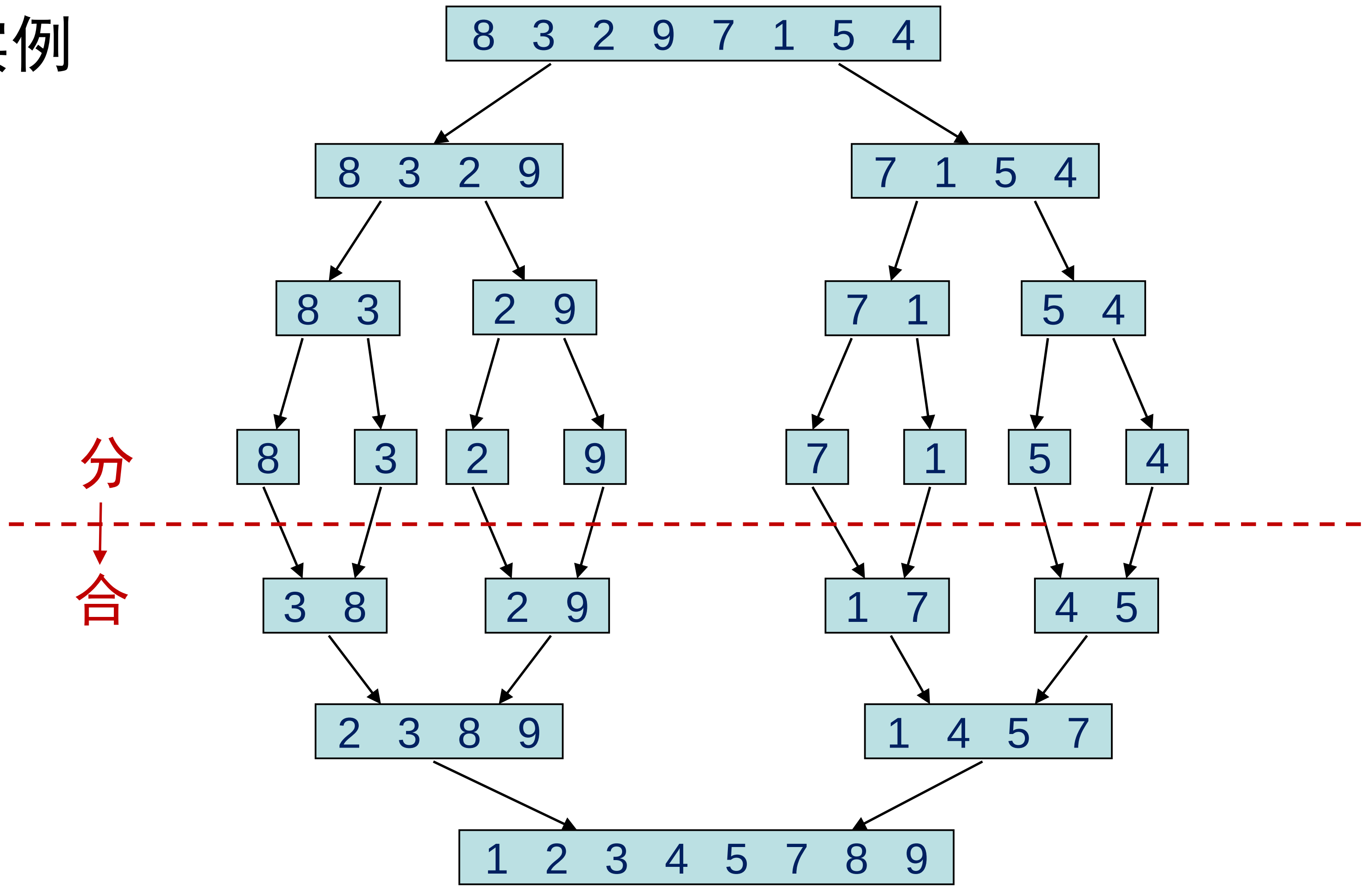
if $i = p$

  copy $C[j..q-1]$ to $A[k..p+q-1]$ ——→ 把某一个未处理完的数组尾部直接复制到输出数组

else copy $B[i..p-1]$ to $A[k..p+q-1]$

合并排序

- 实例


8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
分
合
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 7 8 9

合并排序

- 时间效率
  - 基本操作：比较
  - 假设$n = 2^k$，执行次数的递推式为

$$C(n) = 2C(n/2) + C_{merge}(n),\ \ n > 1$$

$$C(1) = 0$$

    - $C_{merge}(n)$的最差情况：$C_{merge}(n) = n - 1$

  - 最差效率
    - 基于主定理，可推导得：$C_{worst}(n) \in \Theta(n\log n)$

- 额外优势：可实现稳定排序

目录

- 合并排序
- 快速排序
- 二叉树遍历
- 大整数乘法和Strassen矩阵乘法
- 最近对问题和凸包问题

快速排序

- 算法思想：对于输入A[0:n-1]，按以下3个步骤操作
  - 分区（Partition）：以A中某一元素为中轴(Pivot)，将A[0:n-1]划分成3段：A[0:s-1]、A[s]、A[s+1:n-1]，保证

$$\underbrace{A[0]\ldots A[s-1]}_{\text{all are } \le A[s]} \quad A[s] \quad \underbrace{A[s+1]\ldots A[n-1]}_{\text{all are } \ge A[s]}$$

  - 递归求解：递归调用快速排序对A[0:s-1]和A[s+1:n-1]分别进行排序
  - 合并：合并排序后的A[0:s-1]、A[s]、A[s+1:n-1]

快速排序

- 算法伪代码

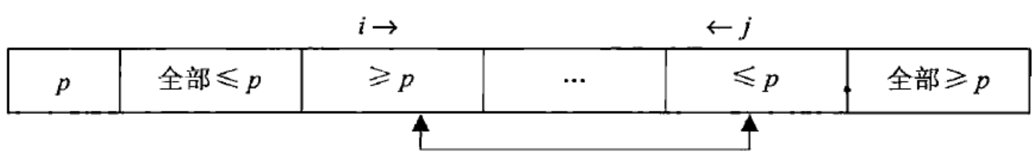
```
算法  Quicksort(A[l..r])
    //用 Quicksort 对子数组排序
    //输入：数组 A[0..n−1]中的子数组 A[l..r]，由左右下标 l 和 r 定义
    //输出：非降序排列的子数组 A[l..r]
    if l < r
        s ← Partition (A[l..r]) //s 是分裂位置
        Quicksort(A[l..s−1])
        Quicksort(A[s+1..r])
```

- 合并排序 vs 快速排序：
  - 前者的子问题划分很直接，主要工作在合并
  - 后者的子问题解的合并很自然，主要工作在划分

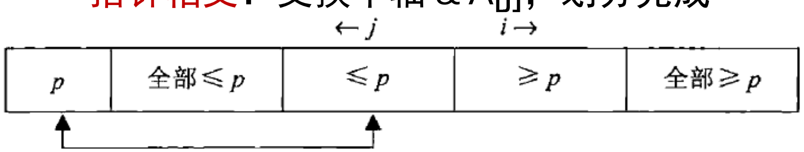
快速排序

- 数组划分算法
  - Lomuto划分（第4章；对数组从左到右扫描）
  - Hoare划分
    - 以数组的第一个元素为中轴，分别从数组的两端进行扫描
      - 指针i从左开始扫描，忽略小于中轴的元素，直到碰到大于或等于中轴的元素A[i]
      - 指针j从右开始扫描，忽略大于中轴的元素，直到碰到小于或等于中轴的元素A[j]

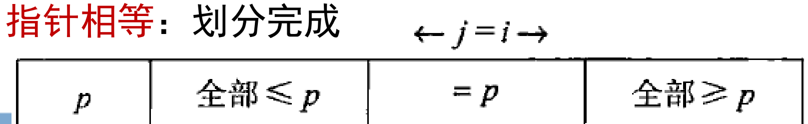
指针不相交：交换A[i] & A[j]，i++, j--，继续扫描



指针相交：交换中轴 & A[j]，划分完成



指针相等：划分完成



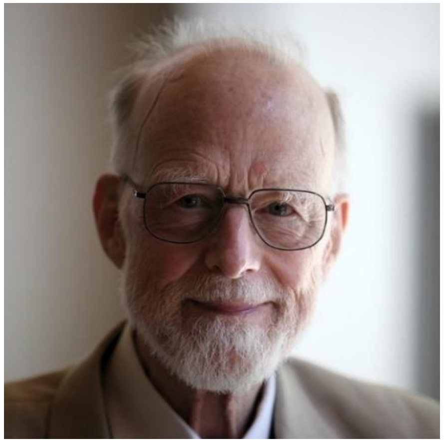
快速排序

- Hoare划分算法伪代码

算法 HoarePartition($A[l..r]$)

//以第一个元素为中轴，对子数组进行划分
//输入：数组 $A[0..n-1]$中的子数组 $A[l..r]$，由左右下标 l 和 r 定义
//输出：$A[l..r]$的一个划分，分裂点的位置作为函数的返回值

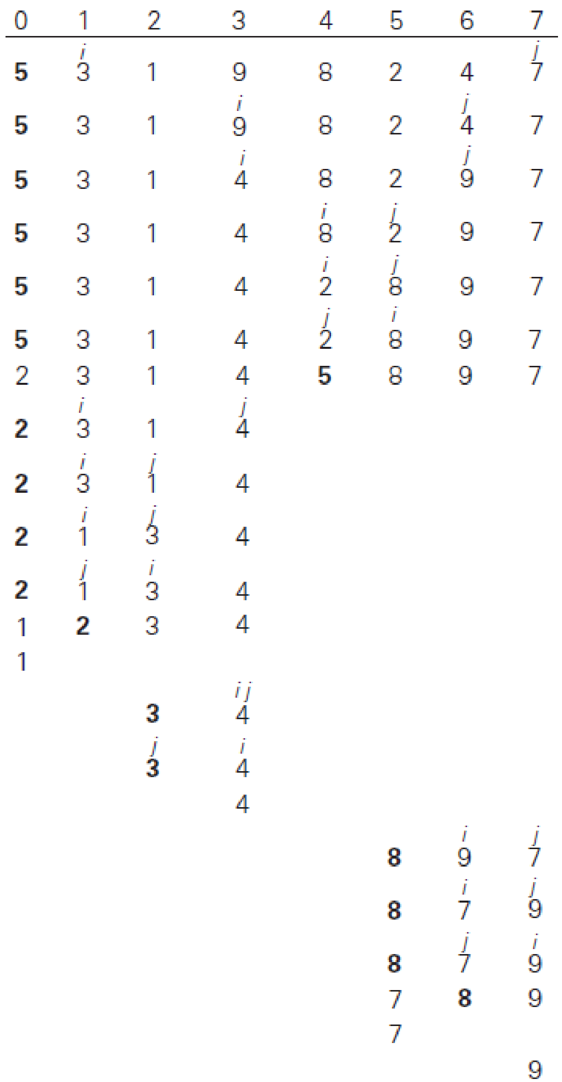
$p \leftarrow A[l]$
$i \leftarrow l;\ j \leftarrow r+1$
repeat
  repeat $i \leftarrow i+1$ **until** $A[i] \geqslant p$
  repeat $j \leftarrow j-1$ **until** $A[j] \leqslant p$
  swap ($A[i], A[j]$)
until $i \geqslant j$
swap ($A[i], A[j]$) //当 $i \geqslant j$ 撤销最后一次交换
swap ($A[l], A[j]$)
return j



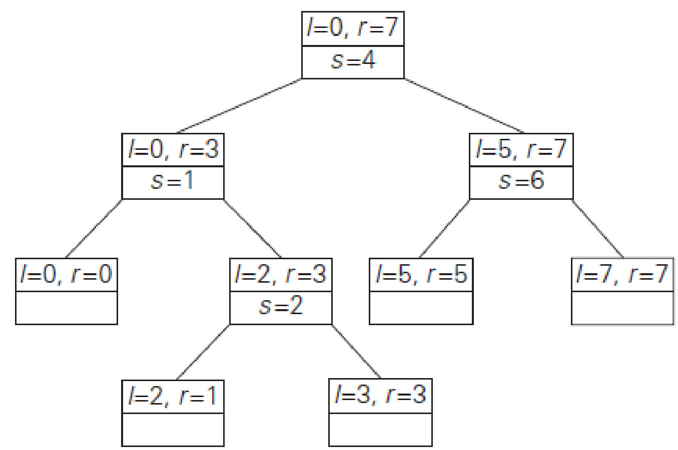
查尔斯·安东尼·理查德·霍尔爵士（Sir **Charles Antony Richard Hoare**，1934年1月11日 - ），昵称为**托尼·霍尔**（英语：**Tony Hoare**），生于大英帝国锡兰可伦坡（今斯里兰卡），英国计算机科学家、图灵奖得主。他设计了快速排序算法、霍尔逻辑、交谈循序程式。

快速排序

- 应用实例



(a) 数组的变化，黑体为中轴元素



(b) 快排的递归调用

快速排序

- 时间效率
  - 最优效率：所有的分裂点位于相应数组的中点，则键值比较的递推式为

$$C_{best}(n) = 2C_{best}(n/2) + n \quad \text{for } n > 1, \quad C_{best}(1) = 0.$$

    - 根据主定理，$C_{best}(n) \in \Theta(n\log n)$
  - 最差效率：输入是排好序的数组，算法每次分裂的两个子数组中总有一个为空

$$C_{worst}(n) = (n+1) + n + \cdots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2).$$

  - 平均效率：分裂点位于每个位置的概率都为1/n

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0. \quad \Longrightarrow \quad C_{avg}(n) \approx 2n \ln n \approx 1.39 n \log_2 n.$$

快速排序

- 可能的改进方法
  - 更好的中轴选择
    - 随机快速排序：使用随机的元素作为中轴
    - 三平均划分法：最左、最右、中间元素的中位数为中轴
  - 划分方法的改进
    - 三路划分
  - 当子数组足够小时（5到15个元素），改用插入排序
- 内在缺陷：不稳定性

未完待续…

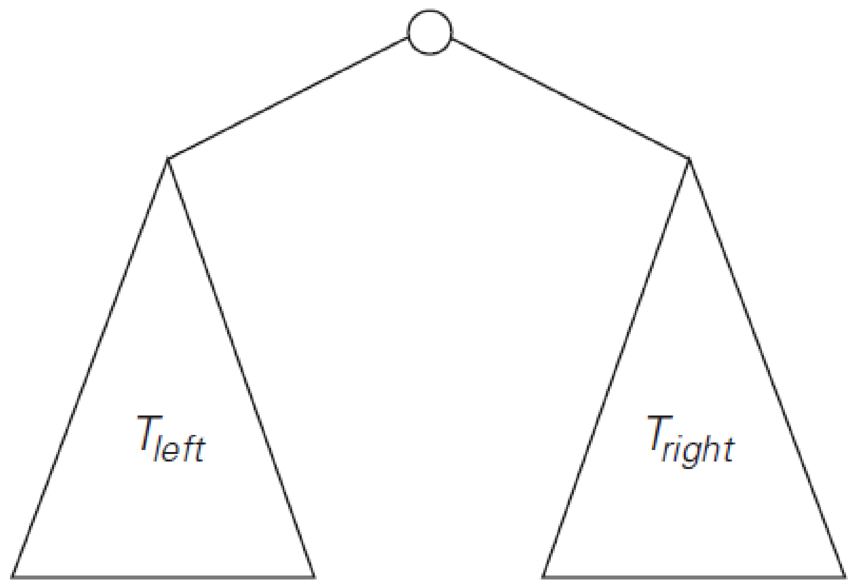
排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

目录

- 合并排序
- 快速排序
- 二叉树遍历
- 大整数乘法和Strassen矩阵乘法
- 最近对问题和凸包问题

二叉树遍历及相关特性

- 二叉树的标准定义



  - 天然适合分治的数据结构

二叉树高度的计算

- 算法

ALGORITHM $Height(T)$

//Computes recursively the height of a binary tree
//Input: A binary tree T
//Output: The height of T
if $T = \varnothing$ **return** -1
else return $\max\{Height(T_{left}), Height(T_{right})\} + 1$

- 加法的执行操作次数

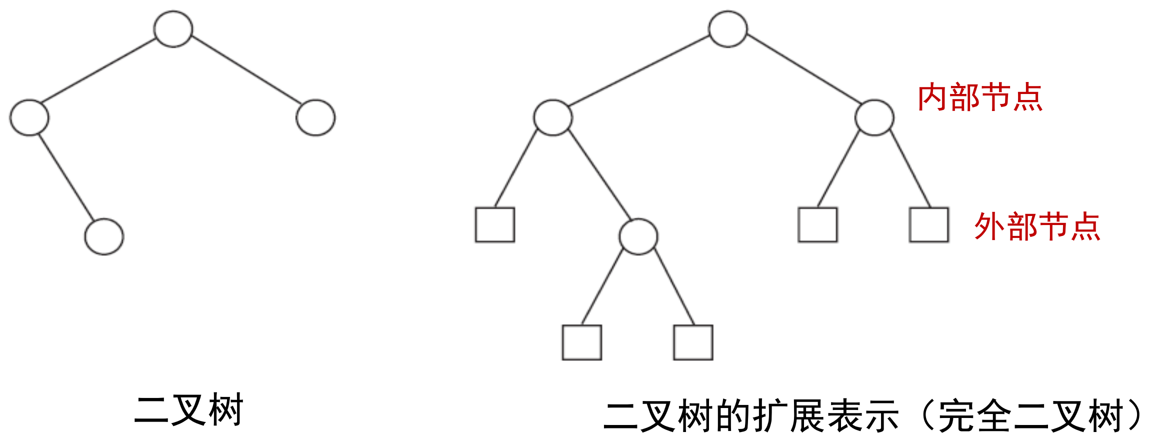
$$A(n(T)) = A(n(T_{left})) + A(n(T_{right})) + 1 \quad \text{for } n(T) > 0,$$

$$A(0) = 0.$$

- 但是算法中，判断T是否为空集的比较操作更频繁

二叉树高度的计算

- 二叉树的扩展表示：增加外部节点，表示空子树



  - 外部节点数x总比内部节点数大1：x=n+1 (why?)
- 检查树是否为空的比较操作执行次数

$$C(n) = n + x = 2n + 1$$

- 而加法的执行操作次数：A(n)=n

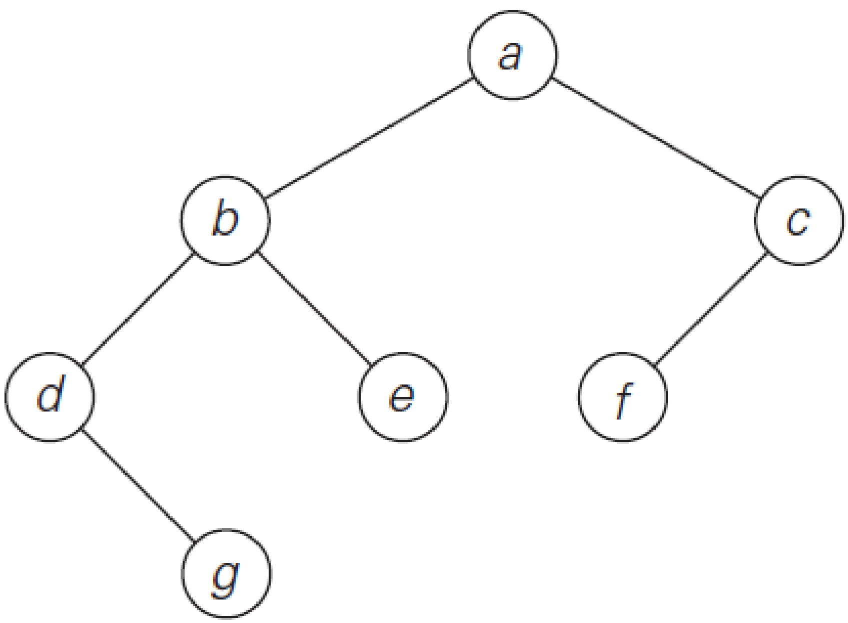
二叉树的遍历

- 遍历：遵循某一种次序来访问二叉树上的所有节点，使得中每一个节点被访问了一次且只访问一次
- 由于二叉树是一种非线性结构，节点可能有不止一个的直接后继节点，所以遍历前必须先规定访问顺序

- 三种经典的遍历算法
  - 前序(preorder)：根->左子树->右子树
  - 中序(inorder)：左子树->根->右子树
  - 后序(postorder)：左子树->右子树->根

二叉树的遍历

- 举例



preorder: *a, b, d, g, e, c, f*
inorder: *d, g, b, e, a, f, c*
postorder: *g, d, e, b, f, c, a*

目录

- 合并排序
- 快速排序
- 二叉树遍历
- 大整数乘法和Strassen矩阵乘法
- 最近对问题和凸包问题

大整数乘法

- 密码技术中，往往需要对超过100位的十进制整数进行乘法运算
- 经典的笔算算法

A = 12345678901357986429　　B = 87654321284820912836

$$
\begin{array}{r}
a_1\ a_2 \ldots\ a_n \\
b_1\ b_2 \ldots\ b_n \\
\hline
(d_{10})\ d_{11}d_{12} \ldots\ d_{1n} \\
(d_{20})\ d_{21}d_{22} \ldots\ d_{2n} \\
\ldots\ \ldots\ \ldots\ \ldots\ \ldots \\
(d_{n0})\ d_{n1}d_{n2} \ldots\ d_{nn}
\end{array}
$$

效率：$O(n^2)$

大整数乘法

- 分治法求解：以两位数的乘法（23x14）为例

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0 \quad \text{and} \quad 14 = 1 \cdot 10^1 + 4 \cdot 10^0.$$

$$\begin{aligned} 23 * 14 &= (2 \cdot 10^1 + 3 \cdot 10^0) * (1 \cdot 10^1 + 4 \cdot 10^0) \\ &= (2 * 1)10^2 + (2 * 4 + 3 * 1)10^1 + (3 * 4)10^0. \end{aligned}$$

  - 上述计算仍需要4次乘法，但是

$$2 * 4 + 3 * 1 = (2 + 3) * (1 + 4) - 2 * 1 - 3 * 4.$$

乘法次数减少到3次

- 任意两位数$a = a_1 a_0$和 $b = b_1 b_0$的乘法

$$c = a * b = c_2 10^2 + c_1 10^1 + c_0$$

$$c_2 = a_1 * b_1 \qquad c_0 = a_0 * b_0$$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

大整数乘法

- 考虑任意n位数的乘法
  - 从中间把每个数字一分为二，表示为

$$a = a_1 a_0 \Longrightarrow a = a_1 10^{n/2} + a_0$$

$$b = b_1 b_0 \Longrightarrow b = b_1 10^{n/2} + b_0.$$

  - 则有

$$\begin{aligned} c = a * b &= (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0) \\ &= (a_1 * b_1) 10^n + (a_1 * b_0 + a_0 * b_1) 10^{n/2} + (a_0 * b_0) \\ &= c_2 10^n + c_1 10^{n/2} + c_0, \end{aligned}$$

$$c_0 = a_0 * b_0 \qquad c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

$$c_2 = a_1 * b_1$$

大整数乘法

- 效率分析
  - 乘法的执行次数

$$M(n) = 3M(n/2) \quad \text{for } n > 1, \quad M(1) = 1.$$

$$\xRightarrow{n = 2^k} M(2^k) = 3M(2^{k-1}) = 3[3M(2^{k-2})] = 3^2 M(2^{k-2})$$

$$= \cdots = 3^i M(2^{k-i}) = \cdots = 3^k M(2^{k-k}) = 3^k$$

$$\Longrightarrow M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}.$$

  - 加减、乘法的总执行次数

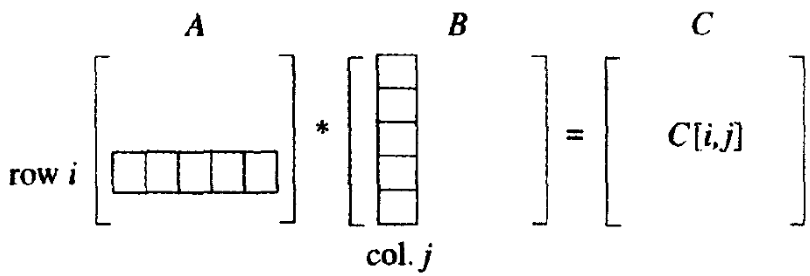
$$A(n) = 3A(n/2) + cn \quad \text{for } n > 1, \quad A(1) = 1.$$

$$\Longrightarrow A(n) \in \Theta(n^{\log_2 3})$$

加减法运算总数与乘法运算次数有相同的渐进增长次数

矩阵乘法

- 传统蛮力法：O(n³) \ Θ(n³)



算法 *MatrixMultiplication*(*A*[0..*n*-1,0..*n*-1],*B*[0..*n*-1,0..*n*-1])

```
//用基于定义的算法计算两个 n 阶矩阵的乘积
//输入：两个 n 阶矩阵 A 和 B
//输出：矩阵 C = AB
for i ← 0 to n-1 do
    for j ← 0 to n - 1 do
        C[i,j] ← 0.0
        for k ← 0 to n -1 do
            C[i,j] ← C[i,j] + A[i,k]*B[k,j]
return C
```

Strassen矩阵乘法

- 分治改进：考虑两个2x2矩阵，我们可以将矩阵乘法表示为

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

其中

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11}),$$
$$m_2 = (a_{10} + a_{11}) * b_{00},$$
$$m_3 = a_{00} * (b_{01} - b_{11}),$$
$$m_4 = a_{11} * (b_{10} - b_{00}),$$
$$m_5 = (a_{00} + a_{01}) * b_{11},$$
$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01}),$$
$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11}).$$

	乘法	加法
蛮力法	8	4
分治改进	7	18

Strassen矩阵乘法

- 分治改进：考虑n x n矩阵维度（$n = 2^k$），我们将矩阵A、B、C分别用4个n/2 x n/2维度的子矩阵表示

$$\left[\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right] = \left[\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right] * \left[\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right]$$

则可通过计算以下7个降维的矩阵乘法来求解C

$$M_1 = (A_{00} + A_{11}) \times (B_{00} + B_{11})$$
$$M_2 = (A_{10} + A_{11}) \times B_{00}$$
$$M_3 = A_{00} \times (B_{01} - B_{11})$$
$$M_4 = A_{11} \times (B_{10} - B_{00})$$
$$M_5 = (A_{00} + A_{01}) \times B_{11}$$
$$M_6 = (A_{10} - A_{00}) \times (B_{00} + B_{01})$$
$$M_7 = (A_{01} - A_{11}) \times (B_{10} + B_{11})$$

$$\Longrightarrow$$

$$C_{00} = M_1 + M_4 - M_5 + M_7$$
$$C_{01} = M_3 + M_5$$
$$C_{10} = M_2 + M_4$$
$$C_{11} = M_1 - M_2 + M_3 + M_6$$

Strassen矩阵乘法

- 算法效率
  - 乘法的执行次数M(n)

$$M(n) = 7M(n/2) \quad \text{for } n > 1, \quad M(1) = 1.$$

Since $n = 2^k$,

$$M(2^k) = 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2 M(2^{k-2}) = \cdots$$
$$= 7^i M(2^{k-i}) \cdots = 7^k M(2^{k-k}) = 7^k.$$

Since $k = \log_2 n$,

$$M(n) = 7^{\log_2 n} = \boxed{n^{\log_2 7} \approx n^{2.807}}.$$

  - 乘加法的执行次数A(n)

$$A(n) = 7A(n/2) + 18(n/2)^2 \quad \text{for } n > 1, \quad A(1) = 0.$$

$$\Longrightarrow A(n) \in \Theta(n^{\log_2 7}) < \Theta(n^3)$$

矩阵乘法

- 持续改进

Name of Method	Complexity
Conventional Method	n^3
Strassen	$n^{2.808}$
Pan	$\omega < 2.796$
Bini	$\omega < 2.78$
Schönhage	$\omega < 2.548$
Romani	$\omega < 2.517$
Coppersmith & Winograd	$\omega < 2.496$
Strassen's New Method	$\omega < 2.479$
Coppersmith & Winograd	$\omega < 2.376$

Open Question：是否存在一种O(n^2)的算法？

目录

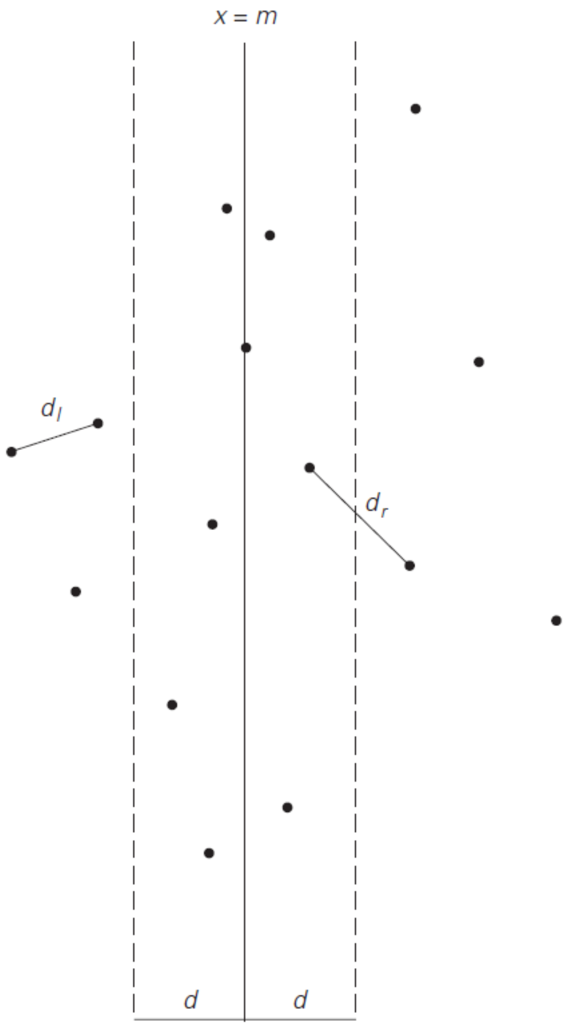
- 合并排序
- 快速排序
- 二叉树遍历
- 大整数乘法和Strassen矩阵乘法
- 最近对问题和凸包问题

最近对问题

- 问题：在包含n个点的平面S中，找出距离最近的两个点

- 算法
  - 蛮力法（n较小时，n≤3）：直接求解
  - 分治改进（n较大时）：
    - 将S上的n个点分成大致相等的2个子集S1和S2
    - 分别求S1和S2中的最近点对
    - 求这两对中距离最近的一对
    - 求一点在S1、另一点在S2中的距离更近的可能点对

最近对问题

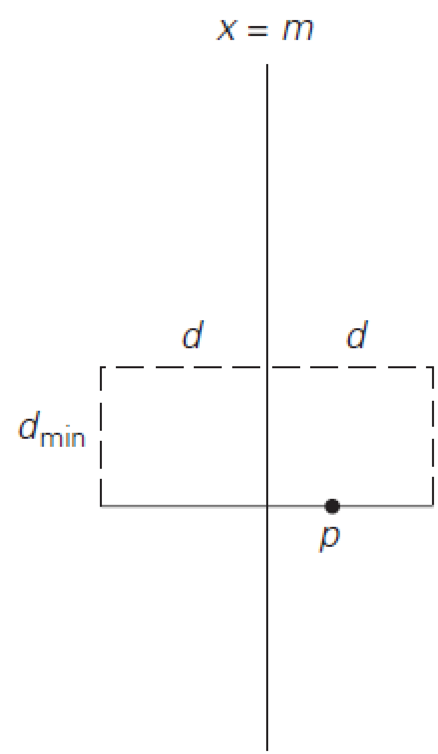


(a)最近对问题的分治算法的思想

$$d = \min\{d_l, d_r\}$$

$$初始 d_{\min} = d$$

$$\Longrightarrow$$



(b)和点 p 距离小于 $d_{\min}$ 的点可能分布的矩形区域

最近对问题

ALGORITHM *EfficientClosestPair*(P, Q)

```
//Solves the closest-pair problem by divide-and-conquer
//Input: An array P of n ≥ 2 points in the Cartesian plane sorted in
//       nondecreasing order of their x coordinates and an array Q of the
//       same points sorted in nondecreasing order of the y coordinates
//Output: Euclidean distance between the closest pair of points
if n ≤ 3
    return the minimal distance found by the brute-force algorithm
else
    copy the first ⌈n/2⌉ points of P to array P_l
    copy the same ⌈n/2⌉ points from Q to array Q_l
    copy the remaining ⌊n/2⌋ points of P to array P_r
    copy the same ⌊n/2⌋ points from Q to array Q_r
    d_l ← EfficientClosestPair(P_l, Q_l)
    d_r ← EfficientClosestPair(P_r, Q_r)
    d ←min{d_l, d_r}
    m ← P[⌈n/2⌉ − 1].x
    copy all the points of Q for which |x − m| < d into array S[0..num − 1]
    dminsq ← d^2
    for i ← 0 to num − 2 do
        k ← i + 1
        while k ≤ num − 1 and (S[k].y − S[i].y)^2 < dminsq
            dminsq ← min((S[k].x − S[i].x)^2+ (S[k].y − S[i].y)^2, dminsq)
            k ← k + 1
return sqrt(dminsq)
```

最近对问题

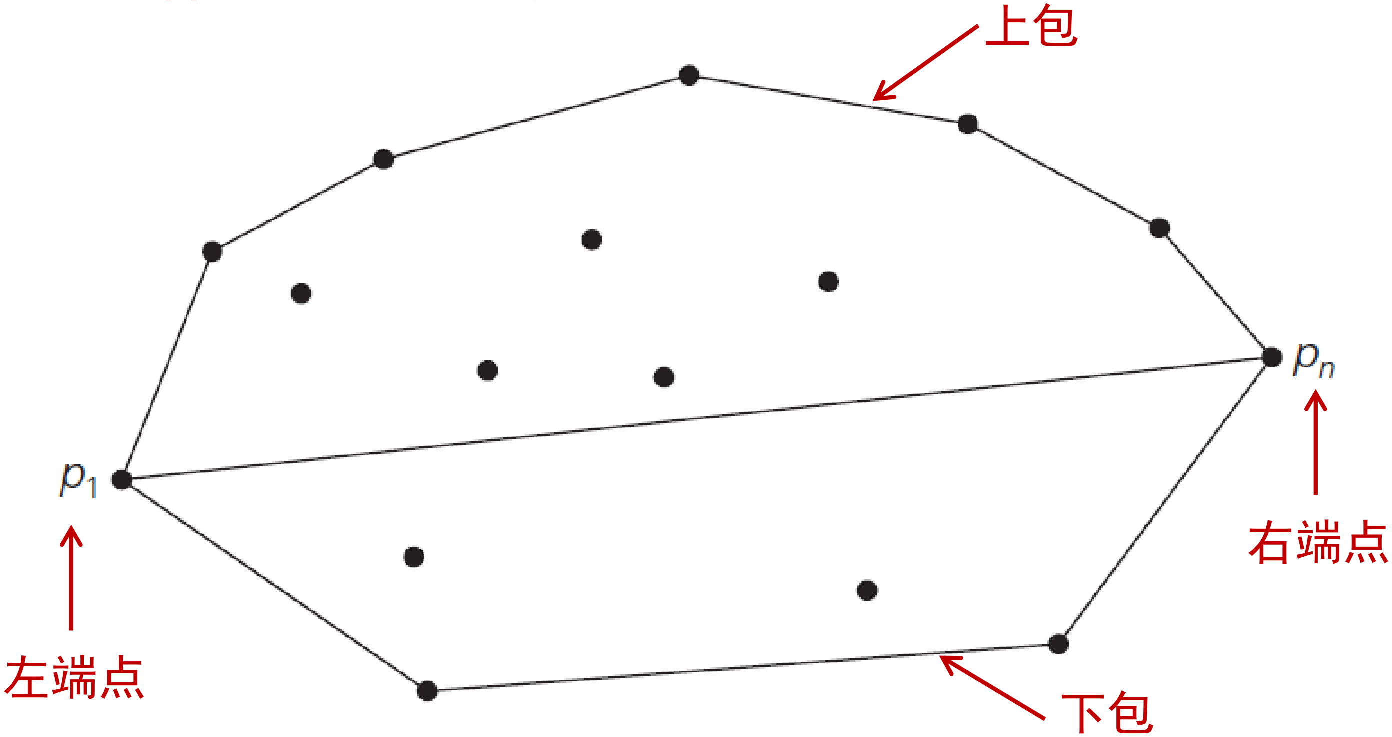
- 算法效率
  - 问题分解和解合并的操作次数是线性复杂度 $f(n) \in \Theta(n)$.
  - 可以证明在解合并过程中，对于近边界的每个S1点，需要考察的S2点最多为6个！
  - 假定$n = 2^k$，则运行时间的递推式为

$$T(n) = 2T(n/2) + f(n) \quad \text{where } f(n) \in \Theta(n).$$

根据主定理，可推得：$T(n) \in \Theta(n \log n)$

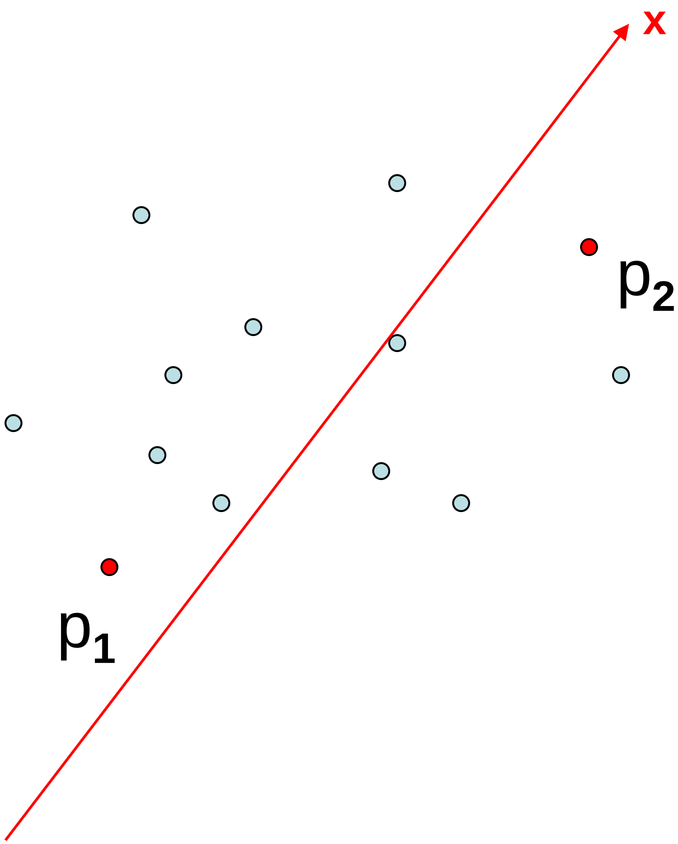
凸包问题

- 问题：给定一个n个点的平面点集S，求S的凸包
- 基于分治思想的快包算法：



凸包问题

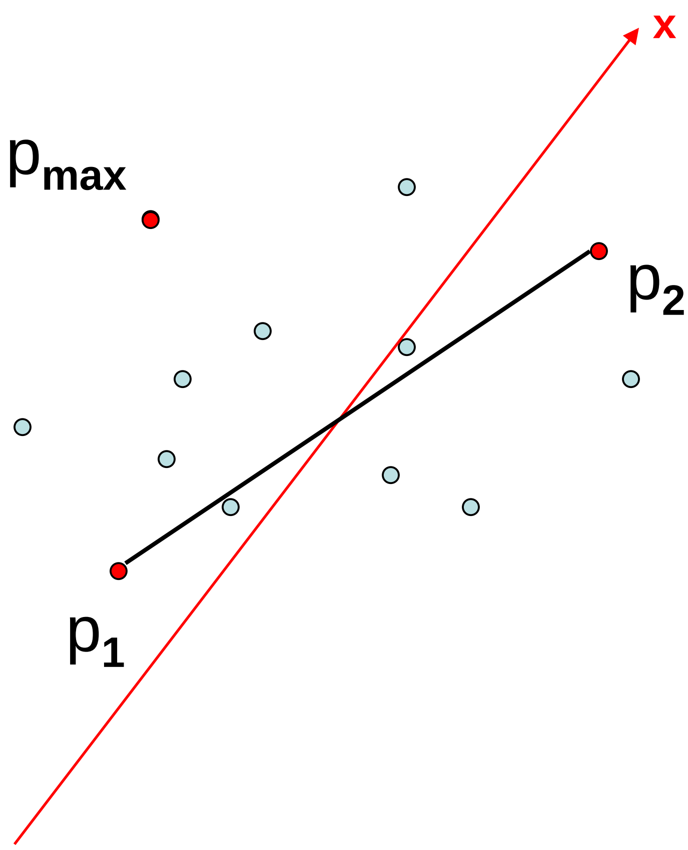
- 快包算法



1、最左边的点p_1和最右边的p_2一定是该集合凸包顶点。

凸包问题

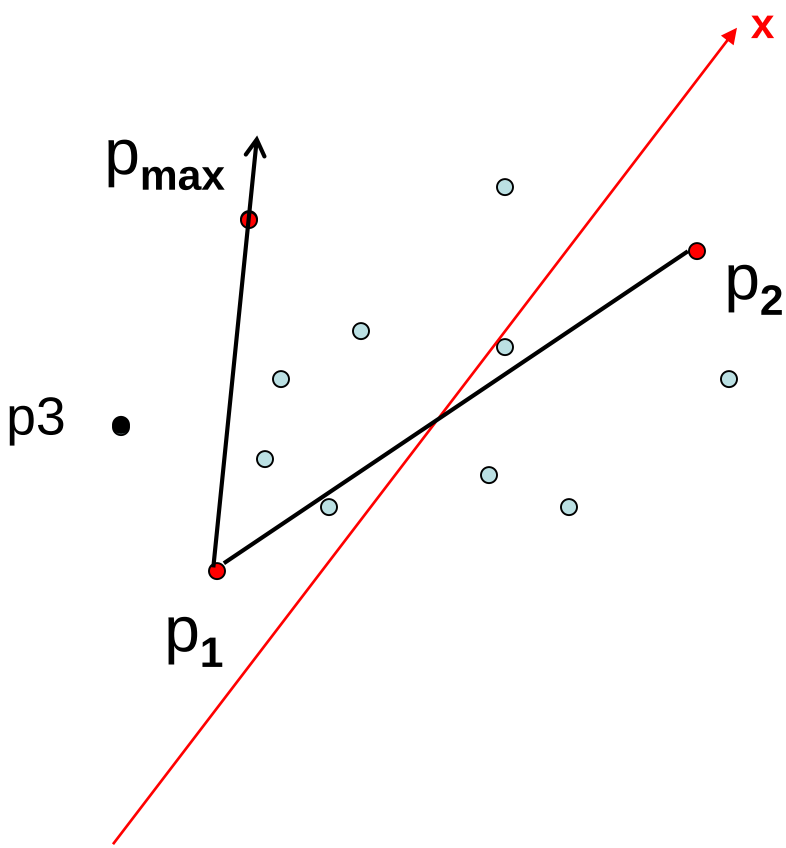
- 快包算法



2、找到上包的顶点，它是距离直线最远的点，如果用两条连接线的话，这个确定了最大的三角形$p_{max}p_1p_2$。

凸包问题

- 快包算法

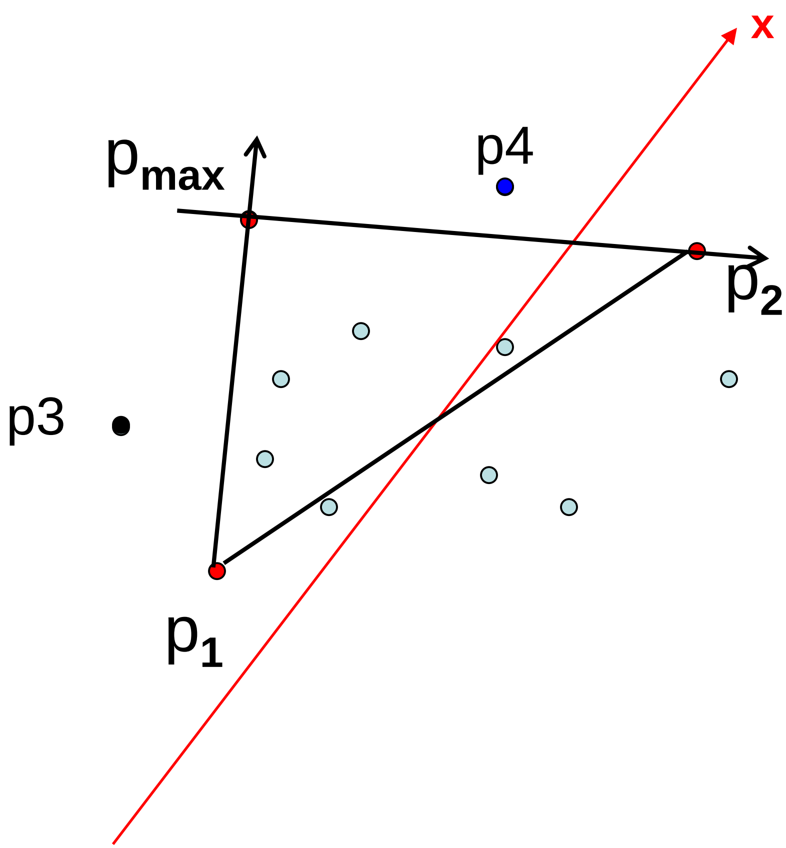


3、找出距离p_1p_{max}左边最远的点p3。

如此进行下去，直到对应的包左边没有点。

凸包问题

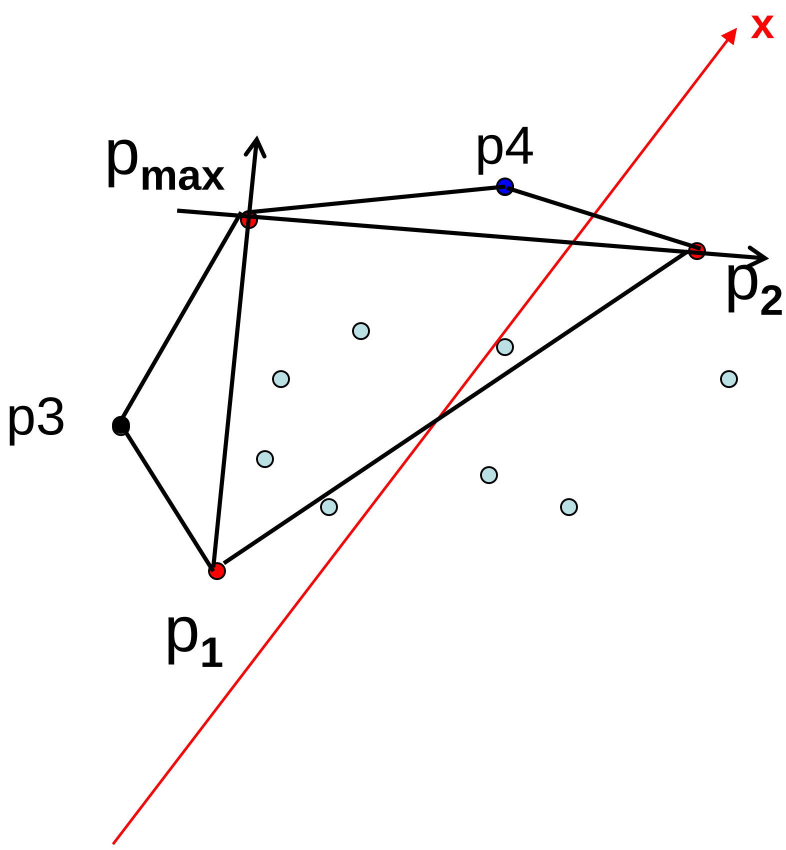
- 快包算法



4、找出$P_{max}P_2$右边最远的点p4，按照该方法进行，直到右边没有点。

凸包问题

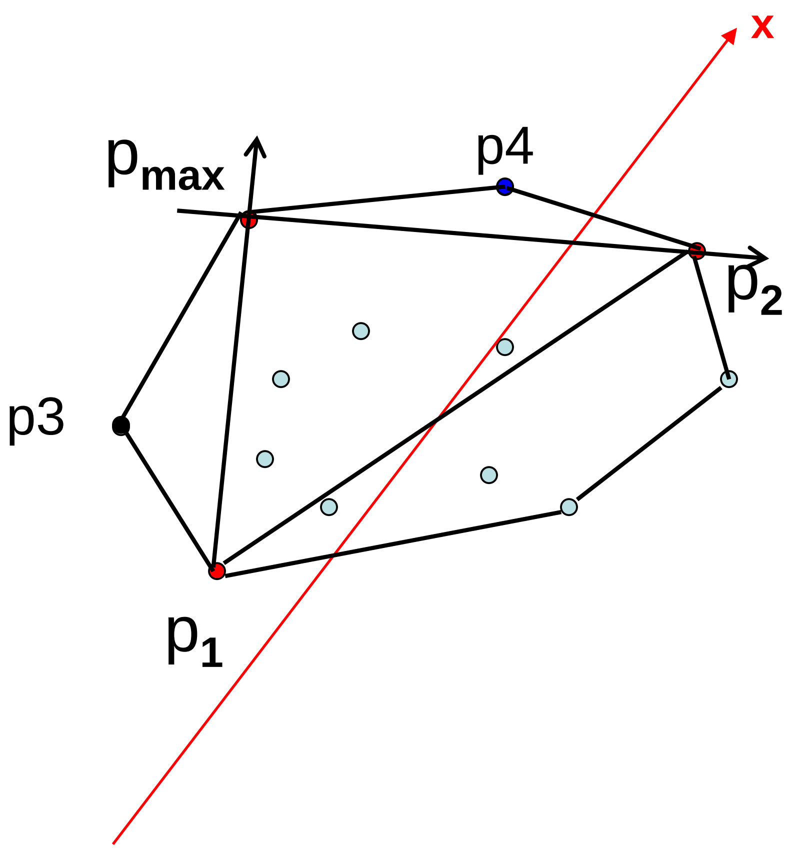
- 快包算法



连接上述这些操作所得到的点，形成上包

凸包问题

- 快包算法



利用上述求上包的方法求出下包

凸包问题

- 如何求距离给定的直线最远的点？
  - 假定平面上有三个点：$q_1(x_1, y_1)$, $q_2(x_2, y_2)$, and $q_3(x_3, y_3)$
  - 所组成的三角形面积等于如下行列式绝对值的1/2：

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

  - 当且仅当 $q_3 = (x_3, y_3)$ 位于直线 $\overrightarrow{q_1q_2}$的左侧时，行列式的值为正
  - 因此，可以用来判断点 $q_3 = (x_3, y_3)$ 位于直线 $\overrightarrow{q_1q_2}$的哪一侧，并且可以求得点到直线的距离

凸包问题

- 算法复杂度
  - 所有点按x轴坐标排序的复杂度：O(nlogn)
  - 找到距离直线$\overrightarrow{q_1q_2}$最远的点：线性时间复杂度
  - 总体时间效率：
    - 最差效率： $\Theta(n^2)$ （跟快排类似）
    - 平均效率： $\Theta(n)$ （给定的点均衡分布在某些凸区域）

分治思想+人工智能（举例）

分布式计算	在大规模数据处理中，可以将数据分解成多个子问题，然后在多个计算节点上并行地解决这些子问题，最后将结果合并起来。这种方法常用于分布式机器学习和大规模数据分析任务中。
集成学习	在集成学习中，可以使用分治法将训练数据集分解成多个子集，然后在每个子集上训练一个基学习器，最后将这些基学习器的结果合并以获得更好的性能。
决策树	决策树是一种基于分治法的分类器，它将特征空间分解成多个子空间，并在每个子空间上构建一个简单的决策模型。通过不断地分解特征空间，决策树可以逐步地将复杂的分类问题分解成简单的子问题。
神经网络分割	在图像分割任务中，可以将图像分解成多个子区域，并在每个子区域上使用神经网络进行分割。最后将这些子区域的分割结果合并以获得整个图像的分割结果。

课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
5	5.1	9	11
	5.2	4, 9	11
	5.3	5	11
	5.4	7	
	5.5	7	11,12

注：只需上交“课后作业题”；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年4月2日