

算法设计与分析



蒋婷婷

上节课回顾

- 递归方程的求解
 - 迭代法
 - 直接迭代
 - 换元迭代
 - 差消化简后迭代
 - 递归树
 - 主定理

主定理

主定理： 设 $a \geq 1, b > 1$ 为常数， $f(n)$ 为函数， $T(n)$ 为非负整数，且

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

则有以下结果：

1. 若 $f(n) = O(n^{\log_b a - \varepsilon}), \varepsilon > 0$ ，那么 $T(n) = \Theta(n^{\log_b a})$
2. 若 $f(n) = \Theta(n^{\log_b a})$ ，那么 $T(n) = \Theta(n^{\log_b a} \log n)$
3. 若 $f(n) = \Omega(n^{\log_b a + \varepsilon}), \varepsilon > 0$ ，且对于某个常数 $c < 1$ 和充分大的 n 有 $a f(n/b) \leq c f(n)$ ，那么 $T(n) = \Theta(f(n))$

顺序算法的设计技术

- 分治策略
- 动态规划算法
- 回溯法与分支估界
- 贪心算法
- 概率算法

分治策略 (Divide and Conquer)

- 分治策略的基本思想
 - 实例、主要思想、算法描述、注意问题
- 递归算法与递推方程
 - 两类递推方程的求解
- 降低递归算法复杂性的途径
 - 代数变换减少子问题个数
 - 预处理减少递归的操作
- 典型实例分析

分治策略的基本思想

分治策略的实例——二分检索、归并排序
主要思想——划分、求解子问题、综合解
算法描述

Divide-and-Conquer(P)

1. if $|P| \leq c$ then $S(P)$.
2. divide P into P_1, P_2, \dots, P_k .
3. for $i = 1$ to k
4. $y_i = \text{Divide-and-Conquer}(P_i)$
5. Return Merge(y_1, y_2, \dots, y_k)

注意问题——连续划分 平衡原则

递归算法与递推方程

- 分治策略的算法分析工具——递推方程
- 两类递推方程

$$f(n) = \sum_{i=1}^k a_i f(n-i) + g(n)$$

$$f(n) = af\left(\frac{n}{b}\right) + d(n)$$

- 求解方法
迭代法、递归树、Master定理

典型的递推方程

$$f(n) = af\left(\frac{n}{b}\right) + d(n)$$

当 $d(n)$ 为常数 时

$$f(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

当 $d(n) = cn$ 时

$$f(n) = \begin{cases} O(n) & a < b \\ O(n \log n) & a = b \\ O(n^{\log_b a}) & a > b \end{cases}$$

实例

例1 芯片测试

A 报告	B 报告	结论
B是好的	A是好的	A,B 都好或 A,B 都坏
B是好的	A是坏的	至少一片是坏的
B是坏的	A是好的	至少一片是坏的
B是坏的	A是坏的	至少一片是坏的

条件：有 n 片芯片，（好芯片至少比坏芯片多1片），

问题：使用最少测试次数，从中挑出1片好芯片

要求：说明测试算法，进行复杂性分析

算法

1. $k \leftarrow n$
2. **while** $k > 3$ **do**
3. 将芯片分成 $\lfloor k/2 \rfloor$ 组
4. **for** $i = 1$ **to** $\lfloor k/2 \rfloor$ **do**
5. **if** 2片好, 则任取1片留下
6. **else** 2片同时丢掉
7. $k \leftarrow$ 剩下的芯片数
8. **if** $k = 3$
9. **then** 任取2片芯片测试
10. **if** 至少1坏, 取没测的芯片
11. **else** 任取1片被测芯片
12. **if** $k = 2$ **or** 1 **then** 任取1片

分析

□ 说明

上述算法只是一个概要说明, 对于 n 为奇数的情况需要进一步处理, 处理时间为 $O(n)$.

□ 复杂性分析

设 $W(n)$ 表示 n 片芯片测试的次数, 则

$$W(n) = W(n/2) + O(n)$$

$$W(1) = 0$$

由Master定理, $W(n) = O(n)$

实例

例2 求一个数的幂

问题：计算 a^n , n 为自然数

传统算法： $\Theta(n)$

分治法

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

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log n).$$

计算 Fibonacci 数

Fibonacci 数的定义

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

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

通常算法：从 F_0, F_1, \dots , 根据定义陆续相加
时间为 $\Theta(n)$

利用数幂乘法的分治算法

定理1 设 $\{F_n\}$ 为 Fibonacci 数构成的数列, 那么

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

证明 : 对 n 进行归纳

算法: 令矩阵 $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, 用分治法计算 M^n

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

提高算法效率的途径1

方法一：代数变换 减少子问题个数

例3 位乘问题

设 X, Y 是两个 n 位二进制数, $n = 2^k$, 求 XY .

传统算法 $W(n)=O(n^2)$

分治法 令 $X = A2^{n/2} + B, Y = C2^{n/2} + D$.

$$XY = AC 2^n + (AD + BC) 2^{n/2} + BD$$

$$W(n) = 4W(n/2) + cn,$$

$$W(1) = 1$$

解得 $W(n) = O(n^{\log_2 4}) = O(n^2)$

代数变换

$$AD + BC = (A - B) (D - C) + AC + BD$$

递推方程

$$W(n) = 3 W(n/2) + cn$$

$$W(1) = 1$$

解

$$W(n) = O(n^{\log 3}) = O(n^{1.59})$$

矩阵乘法

例4 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

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

递推方程 $W(n) = 8 W(n/2) + cn^2$

$$W(1) = 1$$

解 $W(n) = O(n^3)$.

变换方法

$$M_1 = A_{11} (B_{12} - B_{22})$$

$$M_2 = (A_{11} + A_{12}) B_{22}$$

$$M_3 = (A_{21} + A_{22}) B_{11}$$

$$M_4 = A_{22} (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21}) (B_{11} + B_{12})$$

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

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

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

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

Strassen 矩阵乘法

递推方程是

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

$$W(1) = 1$$

由Master定理得

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

提高算法效率的途径2

算法中的处理尽可能提到递归外面作为预处理

例6 平面点对问题

输入：集合 S 中有 n 个点， $n > 1$ ，

输出：所有的点对之间的最小距离。

通常算法： $C(n,2)$ 个点对计算距离，比较最少需 $O(n^2)$ 时间

分治策略：子集 P 中的点划分成两个子集 P_L 和 P_R

$$|P_L| = \left\lceil \frac{|P|}{2} \right\rceil \quad |P_R| = \left\lfloor \frac{|P|}{2} \right\rfloor$$

平面最近点对算法

MinDistance(P, X, Y)

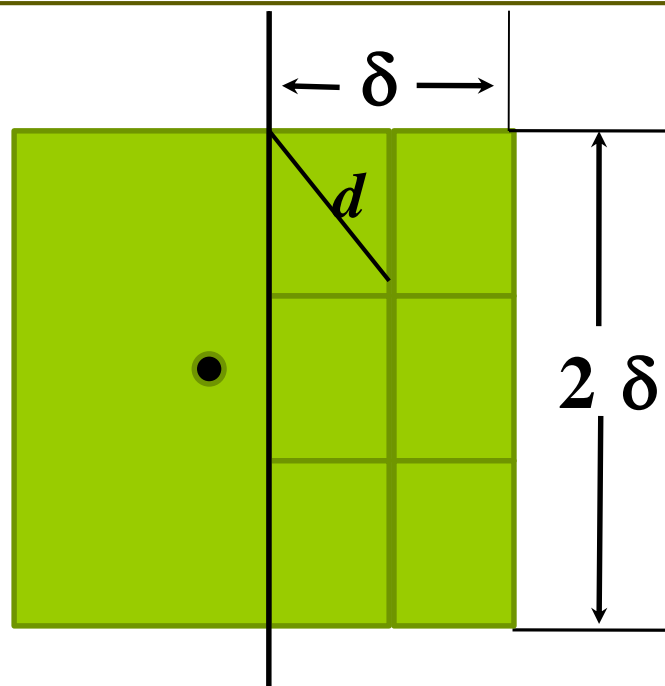
输入: n 个点的集合 P , X 和 Y 分别为横、纵坐标数组

输出: 最近的两个点及距离

1. 如果 P 中点数小于等于3, 则直接计算其中的最小距离
2. 排序 X, Y
3. 做垂直线 l 将 P 划分为 P_L 和 P_R , P_L 的点在 l 左边, P_R 的点在 l 右边
4. **MinDistance(P_L, X_L, Y_L); $\delta_L = P_L$ 中的最小距离**
5. **MinDistance(P_R, X_R, Y_R); $\delta_R = P_R$ 中的最小距离**
6. **$\delta = \min (\delta_L, \delta_R)$**
7. 对于在垂直线两边距离 δ 范围内的每个点, 检查是否有
点与它的距离小于 δ , 如果存在则将 δ 修改为新值

跨边界的最近点

$$\begin{aligned}d &= \sqrt{(\delta / 2)^2 + (2\delta / 3)^2} \\&= \sqrt{\delta^2 / 4 + 4\delta^2 / 9} \\&= \sqrt{25\delta^2 / 36} = 5\delta / 6\end{aligned}$$



右边每个小方格至多1个点，每个点至多比较对面的6个点，只需考察常数个点。将边界区域内的点按照纵坐标进行扫描，对于每个点进行检查，考察在另一侧相关区域内的点（不超过6个），检查1个点是常数时间， $O(n)$ 个点需要 $O(n)$ 时间

算法分析

分析：步1 $O(1)$
步2 $O(n \log n)$
步3 $O(1)$
步4-5 $2T(n/2)$
步6 $O(1)$
步7 $O(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

$$T(n) = O(1) \quad n \leq 3$$

由递归树估计 $T(n) = O(n \log^2 n)$

预排序的处理方法

在每次调用时将已经排好的数组分成两个排序的子集，
每次调用这个过程的时间为 $O(n)$

$W(n)$ 总时间， $T(n)$ 算法递归过程， $O(n\log n)$ 预处理排序

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

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

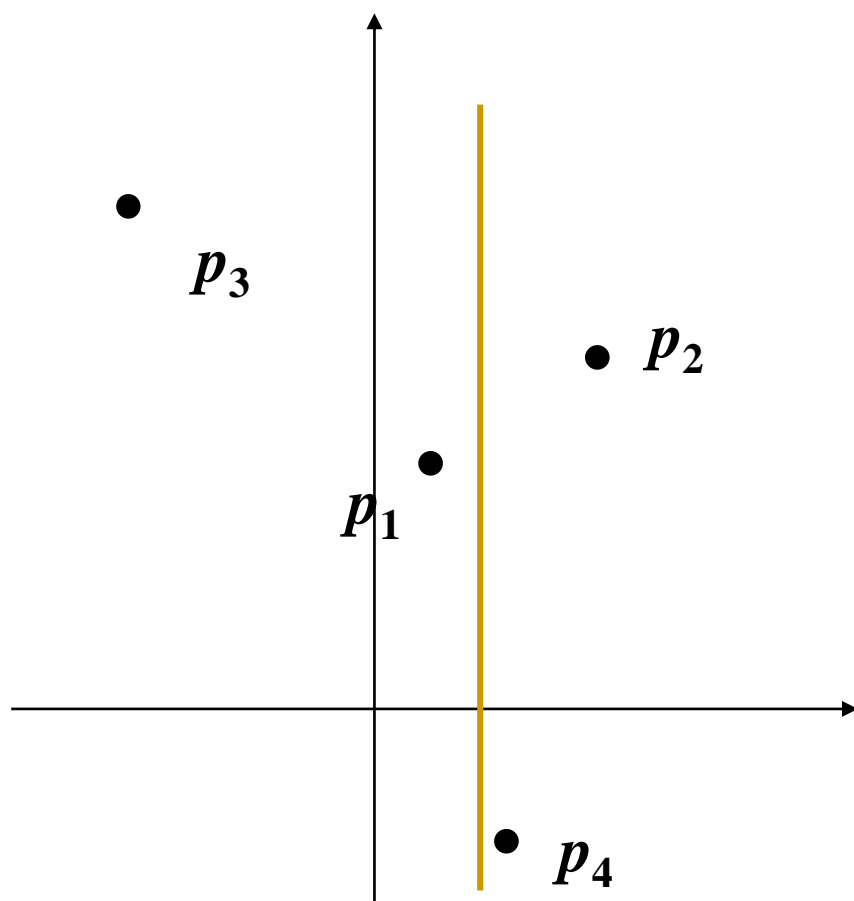
$$T(n) = O(1) \quad n \leq 3$$

解得

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

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

实例：递归中的拆分



P	1	2	3	4
x	0.5	2	-2	1
y	2	3	4	-1

X	-2(3)	0.5(1)	1(4)	2(2)
Y	-1(4)	2(1)	3(2)	4(3)

X_L	-2(3)	0.5(1)
X_R	1(4)	2(2)
Y_L	2(1)	4(3)
Y_R	-1(4)	3(2)

典型实例分析

算法 快速排序

输入：数组 $A[p..r]$

输出：排好序的数组 A

Quicksort(A, p, r)

1. if $p < r$
2. then $q \leftarrow \text{Partition}(A, p, r)$
3. $A[p] \leftrightarrow A[q]$
4. **Quicksort**($A, p, q-1$)
5. **Quicksort**($A, q+1, r$)

划分过程

Partition(A, p, r)

1. $x \leftarrow A[p]$
2. $i \leftarrow p$
3. $j \leftarrow r+1$
4. **while true do**
5. **repeat** $j \leftarrow j - 1$
6. **until** $A[j] \leq x$
7. **repeat** $i \leftarrow i + 1$
8. **until** $A[i] > x$
9. **if** $i < j$
10. **then** $A[i] \leftrightarrow A[j]$
11. **else return** j

实例

27	99	0	8	13	64	86	16	7	10	88	25	90
	i										j	

27	25	0	8	13	64	86	16	7	10	88	99	90
					i				j			

27	25	0	8	13	10	86	16	7	64	88	99	90
						i		j				

27	25	0	8	13	10	7	16	86	64	88	99	90
							j	i				

16	25	0	8	13	10	7	27	86	64	88	99	90
----	----	---	---	----	----	---	----	----	----	----	----	----

复杂度分析

最坏情况 $W(n) = W(n - 1) + O(n)$

$$W(1) = 0$$

$$W(n) = \frac{1}{2} n(n - 1) = \Theta(n^2)$$

最好划分 $T(n) = 2T(\frac{n}{2}) + O(n)$

$$T(1) = 0$$

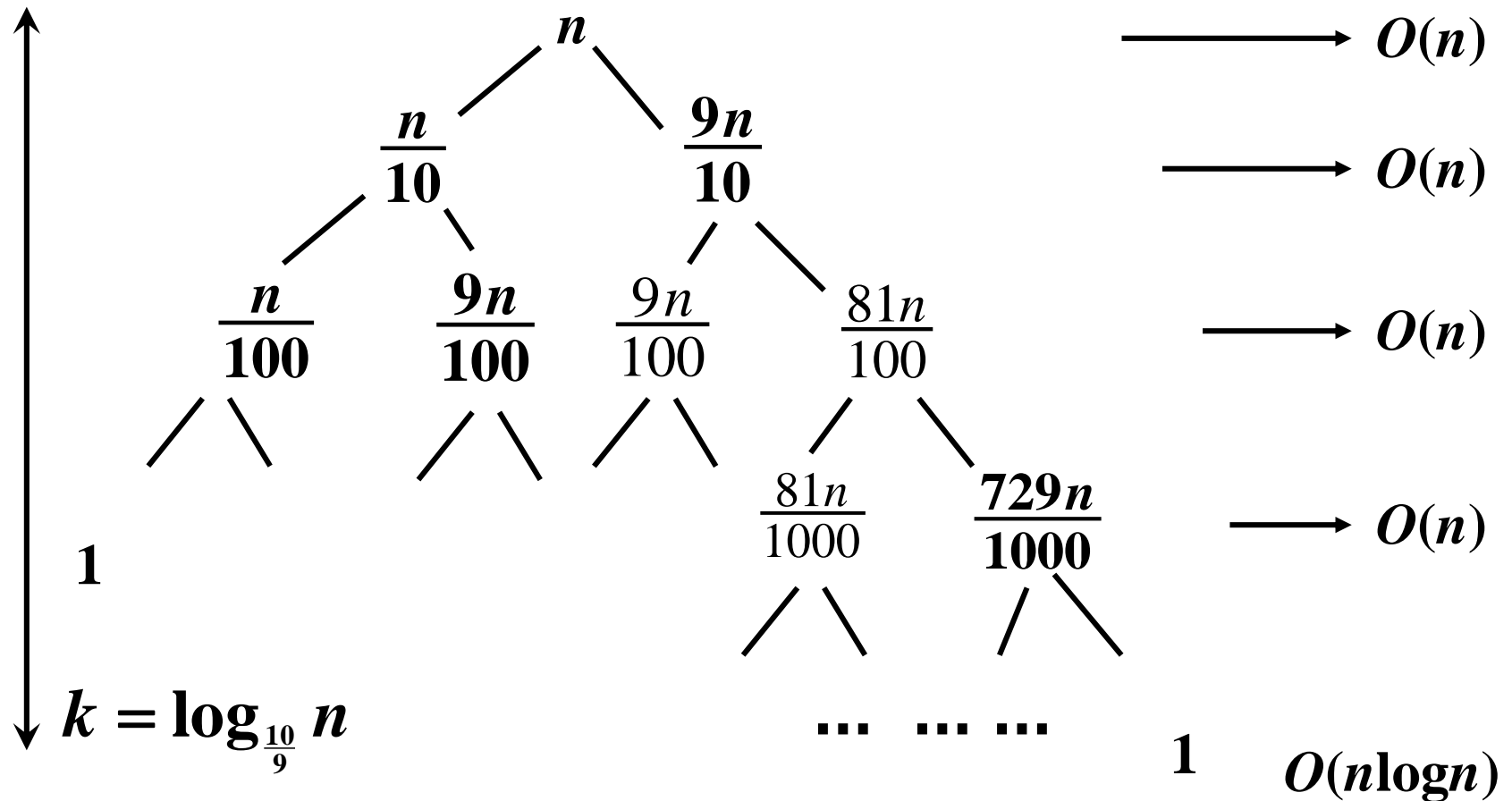
$$T(n) = \Theta(n \log n)$$

均衡划分 $T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + O(n)$

$$T(1) = 0$$

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

均衡划分



平均情况

假设输入数组首元素排好序后的正确位置处在 $1, 2, \dots, n$ 各种情况是等可能的

$$T(n) = \frac{1}{n} \sum_{k=0}^{n-1} (T(k) + T(n - k - 1)) + O(n)$$

$$T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + O(n)$$

$$T(1) = 0$$

利用差消法求得 $T(n)=O(n\log n)$