中国科学技术大学

算法基础

第二讲:渐近记号与递归方程

主 讲: 顾乃杰 教授

单位: 计算机科学技术学院

学期: 2015-2016(秋)







3. Growth of Functions

- 算法运行时间增长的量级很好地刻划了算法的效率,展现了算法的相对性能。
- 对于足够大的输入规模,由于输入规模的影响,常数和低阶项可以忽略。
- When input size is large enough, we concern with how the running time of an algorithm increases with the size of the input *in the limit*, i.e. studying the *asymptotic* efficiency of algorithms.
- Usually, an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.







3.1 渐近记号Asymptotic notation

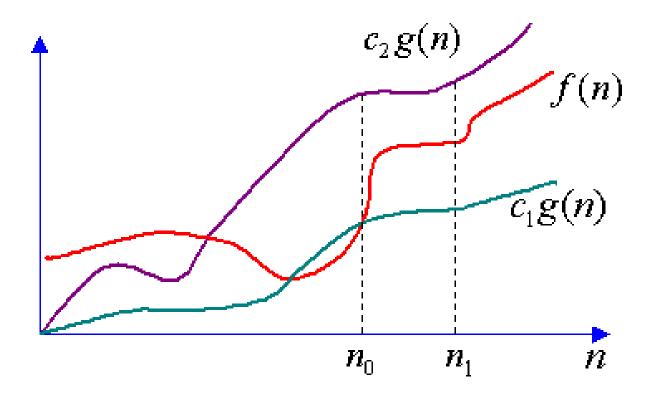
Θ -notation:

定义: $f(n)=\Theta(g(n))$ 意味着存在正常数 C_1 , C_2 和 n_0 使得当 $n \ge n_0$, 均有 $0 \le C_1 g(n) \le f(n) \le C_2 g(n)$ 成立。

- $f(n)=\Theta(g(n))$ 表明,当 $n\to\infty$ 时, f(n) 和g(n)趋于无 穷大的阶是相同的。
- f(n) 和g(n)均是渐近非负的,即:对足够大的,f(n) 和g(n)均大于等于 $\mathbf{0}$ 。



The curves for $f(n) = \Theta(g(n))$



图中的 n_0 点的精确值一般难以得到,在分析和证明过程中可以用一个大于 n_0 而且易于求得的值 n_1 替代。

School of CST



Example: $n^2 / 2 - 3n = \Theta(n^2)$



Proof: We need to chose positive constants C_1 , C_2 , and n_0 , such that $C_1n^2 \le n^2/2 - 3n \le C_2n^2$.

To do so, we let $C_1=1/6$, $C_2=1$, $n_0=9$:

Since $n^2/3-3n = (n/3-3)n \ge 0$, for $n \ge 9$

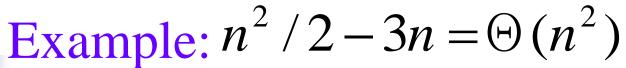
Then $n^2/6 \le n^2/6 + (n^2/3 - 3n) = n^2/2 - 3n$, for $n \ge 9$

And $n^2/2 - 3n \le n^2$, for $n \ge 0$

So $C_1 n^2 \le n^2/2 - 3n \le C_2 n^2$ holds for all $n \ge 9$.









在证明中 C_1 , C_2 和 n_0 的选取可以有多种不同的方式,可取各种不同的值,例如在上题中可以取 C_1 =1/4, C_2 =1/2, n_0 =12, 也同样可以证明。

School of CST



Example: $an^3 + bn^2 + cn + d = \Theta(n^3)$ a > 0

Proof: We Choose
$$n_0 = \lceil 2(|b| + |c| + |d|)/a + 1 \rceil$$
, $c_1 = a/2$
 $c_2 = a + |b| + |c| + |d|$, If $n \ge n_0$, we have:
 $an^3 + bn^2 + cn + d = an^3/2 + (an^3/2 + bn^2 + cn + d)$
 $\ge an^3/2 + (an^3/2 - |b|n^2 - |c|n - |d|)$
 $\ge an^3/2 + (an/2 - |b| - |c| - |d|)n^2$
 $\ge an^3/2 + (a \cdot \lceil 2(|b| + |c| + |d|)/a + 1 \rceil / 2 - |b| - |c| - |d|)n^2$
 $\ge an^3/2 = c_1 n^3$
 $an^3 + bn^2 + cn + d \le an^3 + |b|n^2 + |c|n + |d|) \le c_2 n^3$
So, $c_1 n^3 \le an^3 + bn^2 + cn + d \le c_2 n^3$ holds for all $n \ge n_0$

算法基础





O-notation



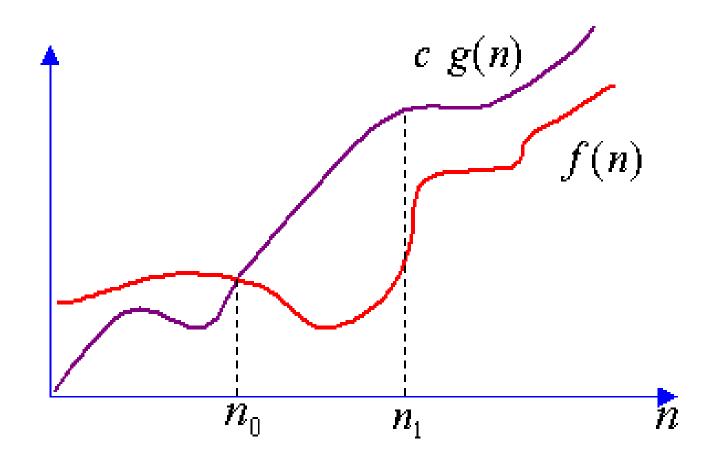
- **定义**: f(n)=O(g(n)) 意味着存在正常数 C 和 n_0 , 使得 当 $n \ge n_0$, 均有 $0 \le f(n) \le C g(n)$ 成立。
- f(n)=O(g(n))表明,当 $n\to\infty$ 时, f(n)趋于无穷大的 阶不大于(即小于等于) g(n)趋于无穷大的阶.
- 若f(n)= $\Theta(g(n))$,则一定有f(n)=O(g(n))







The curves for f(n) = O(g(n))



School of CST





Examples for O-notation

1)
$$an^3 + bn^2 + cn + d = O(n^3)$$
, $a > 0$

Proof: 前面的例子已经表明 $an^3 + bn^2 + cn + d = \Theta(n^3)$

2)
$$an^2 + bn + d = O(n^3)$$
, $a > 0$

Proof: We let c = |a| + |b| + |d|, $n_0 = 1$, then:

$$an^{2} + bn + d \le |a|n^{2} + |b|n + d \le (|a| + |b| + |c|)n^{3} = cn^{3}$$

So, $an^{2} + bn + d = O(n^{3})$

School of CST



Examples of functions in $O(n^2)$

n^2
$n^2 + n$
$n^2 + 1000n$
$1000n^2 + 1000n$

```
n = n / 1000
n^{1.99999}
n^2 / \lg \lg \lg n
```





Ω -notation



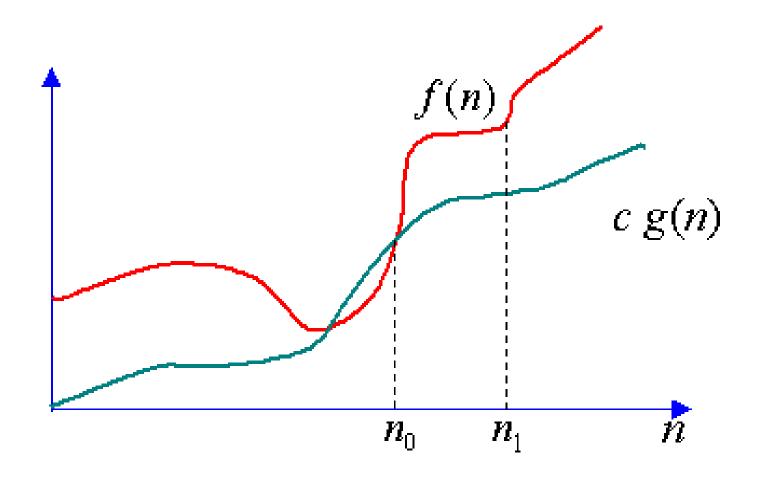
- **定义:** $f(n) = \Omega(g(n))$ 意味着存在正常数 C 和 n_0 ,使得 当 $n \ge n_0$,均有 $0 \le C g(n) \le f(n)$ 成立。
- $f(n)=\Omega(g(n))$ 表明,当 $n\to\infty$ 时, f(n)趋于无穷大的 阶不小于 g(n) 趋于无穷大的阶.







The curves for $f(n) = \Omega(g(n))$





Example: $an^{3} + bn^{2} + d = \Omega(n^{2})$ a > 0



• Proof: Let c = a, $n_0 \ge \frac{2(|b| + |d|)}{1} + 1$, If $n \ge n_0$, we have:

$$\frac{a}{2}n^{3} + bn^{2} + d \ge \frac{a}{2}n^{2} \cdot (\frac{2(|b| + |d|)}{a} + 1) + bn^{2} + d$$

$$\ge n^{2} \cdot (|b| + |d|) + bn^{2} + d + \frac{a}{2}n^{2} \ge \frac{a}{2}n^{2} \qquad (\because n \ge 1)$$

$$an^{3} + bn^{2} + d \ge \frac{a}{2}n^{3} + \frac{a}{2}n^{2} \ge an^{2} = cn^{2}$$

在证明过程中,我们可以让和选取不同的值也同样可以证明。



Example 3.6: $an^3 + bn^2 + d = \Omega(2n^2 + 3n\log n)$

Proof: Let
$$c = a/2$$
, $n_0 \ge \frac{2(|b|+|d|)}{a} + 5$, If $n \ge n_0$, we have:
$$an^3 + bn^2 + d \ge \frac{a}{2}n^3 + (\frac{a}{2}n^3 + bn^2 + d)$$

$$\geq \frac{a}{2}n^3 + (\frac{a}{2}n^2 \frac{2(|b|+|d|)}{a} + bn^2 + d) \geq \frac{a}{2}n^3$$

$$\geq \frac{a}{2} \cdot 5n^2 \geq \frac{a}{2} \cdot 2n^2 + \frac{a}{2} \cdot 3n^2 \geq \frac{a}{2} \cdot 2n^2 + \frac{a}{2} \cdot 3n \log n$$

$$= \frac{a}{2} \cdot (2n^2 + 3n \log n) = c \cdot (2n^2 + 3n \log n)$$

算法基础





Examples of functions in $\Omega(n^2)$

n^2	
$n^{2} + n$	
$n^2 - n$	
$1000n^2 +$	1000n
$1000n^2$ –	1000n

$$n^{3}$$

$$n^{2.00001}$$

$$n^{2} \lg \lg \lg n$$

$$2^{2^{n}}$$



NIVERSITY OF SCIENCE & TECHNOLOGY OF CHINA

Insertion Sort 运行时间

- The best-case running time of insertion sort is $\Omega(n)$
- The worst-case running time of insertion sort is $O(n^2)$
- The running time of insertion sort is between $\Omega(n)$ and $O(n^2)$
- When we say the *running time* of an algorithm is $\Omega(g(n))$, we mean that *no matter what particular input of size n is chosen for each value of n*, the running time on that input is at least a constant times g(n), for sufficiently large n.





NIVERS OFSCIENCE & TE OF

o-notation

- 定义: f(n) = o(g(n))意味着对任意正常数 c ,存在正常数 n_0 使得对所有 $n \ge n_0$, $0 \le f(n) < cg(n)$ 成立。
- O符号和 o 符号的主要区别在于: 在 f(n) = O(g(n)) 的 定义中, $0 \le f(n) \le cg(n)$ 对某些正常数 c 成立, 而在 f(n) = o(g(n)) 的定义中, $0 \le f(n) < cg(n)$ 对所有正常数 c 成立。
- Example 3.8: $15n = o(n \log n)$ $2\log^2 n + d = o(n^{1/2})$





ω -notation



- **定义**: $f(n) = \omega(g(n))$ 意味着对任意正常数 c ,存在正常数 n_0 ,使得对所有 $n \ge n_0$, $0 \le cg(n) < f(n)$ 成立。
- Ω 符号和 ω 符号的主要区别在于: 在 $f(n) = \Omega$ (g(n))的 定义中, $0 \le cg(n) \le f(n)$ 对某些正常数 C 成立, 而在 $f(n) = \omega(g(n))$ 的定义中, $0 \le cg(n) < f(n)$ 对所有正常 C 数成立。
- Example 3.9: $2n^2 + bn + d = \omega(n)$, $\pi = 3n^2 \neq \omega(n^2)$



Asymptotic notation in equations

When on right-hand side:

 $O(n^2)$ stands for some anonymous function in the set $O(n^2)$.

如:
$$2n^2+3n+1=2n^2+\Theta(n)$$
 意味着对某些 $f(n) \in \Theta(n)$, 公式 $2n^2+3n+1=2n^2+f(n)$ 成立. 比如, $f(n)=3n+1$ 时成立.

By the way, we interpret # of anonymous functions as = # of times the asymptotic notation appears:

•
$$\sum_{i=1}^{n} O(i)$$
 OK: 1 anonymous function

• $O(1) + O(2) + \cdots + O(n)$ not OK: n hidden constants \Rightarrow no clean interpretation



Asymptotic notation in equations

When on left-hand side:

No matter how the anonymous functions are chosen on the left-hand side, there is a way to choose the anonymous functions on the right hand side to make the equation valid.

- Interpret $2n^2 + \Theta(n) = \Theta(n^2)$ as meaning for all functions $f(n) \in \Theta(n)$, there exists a function $g(n) \in \Theta(n^2)$ such that $2n^2 + f(n) = g(n)$.
- Can chain together: $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$.

School of CST



渐近关系的传递性: Transitivity

- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ imply $f(n) = \Theta(h(n))$,
- f(n) = O(g(n)) and g(n) = O(h(n)) imply f(n) = O(h(n)),
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$,
- f(n) = o(g(n)) and g(n) = o(h(n)) imply f(n) = o(h(n)),
- $f(n) = \omega(g(n))$ and $g(n) = \omega(h(n))$ imply $f(n) = \omega(h(n))$.





渐近关系的自反性: Reflexivity

$$f(n) = \Theta(f(n)),$$

$$f(n) = O(f(n)),$$

$$f(n) = \Omega(f(n)).$$







对称性 & 置换对称性

• $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

- f(n) = O(g(n)) if and only if $g(n) = \Omega(f(n))$
- f(n) = o(g(n)) if and only if $g(n) = \omega(f(n))$
- 注意:不是任意两个函数都是渐近可比的,存在函数f(n) 和 g(n),使得f(n)=O(g(n)) 不成立,且 $f(n)=\Omega(g(n))$ 也不成立。如: n 和 $n^{1+\sin n}$





Homework 3.1:

■ Page 31, 3.1-2, 3.1-4, 3.1-6





3.2 常见函数

■ 上取整和下取整— Ceilings and Floors

 $\begin{bmatrix} x \end{bmatrix}$ is the least integer greater than or equal to x

|x| is the greatest integer less than or equal to x

For all real
$$x$$
: $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$

For any integer
$$n$$
: $\left| \frac{n}{2} \right| + \left| \frac{n}{2} \right| = n$





上取整和下取整 相关公式

$$\lceil a/b \rceil \le (a+(b-1))/b$$

$$|a/b| \ge (a - (b-1))/b$$



补充定理



f(x) 是任何连续单调上升函数,且f(x) 在整数点才可能取整数值,则:

$$|f(x)| = |f(x)|$$





模运算: Modular arithmetic

$$a \mod n = a - \lfloor a/n \rfloor n$$

 $a \equiv b \pmod{n} \mod n \pmod{n} = (b \mod n)$



指数: Exponentials

$$e^{x} = 1 + x + \frac{x^{2}}{2!} + \frac{x^{3}}{3!} + \dots = \sum_{ki=0}^{\infty} \frac{x^{k}}{k!}$$
 $e^{x} \ge 1 + x$

$$1 + x \le e^x \le 1 + x + x^2$$

$$e^{x} = 1 + x + \Theta (x^{2})$$

$$\lim_{n\to\infty} (1+\frac{x}{n})^n = e^x$$





对数: Logarithms

We shall use the following notations:

$$\log n = \lg n = \log_2 n$$

$$\ln n = \log_e n$$

$$\log^k n = (\log n)^k$$

$$\log \log n = \log(\log n) = \log^{(2)} n$$





阶乘 和 Stirling 公式

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta(\frac{1}{n})\right)$$

$$n! = o(n^n)$$
 $n! = \omega(2^n)$

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





函数迭代: Functional iteration

• For integers $i \ge 0$, recursively define:

$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0\\ f(f^{(i-1)}(n)) & \text{if } i > 0 \end{cases}$$

 $\log^* n$ (read "log star of n") is the iterated logarithm, which is defined as follows:

$$\log^* n = \min\{i \ge 0 : \log^{(i)} n \le 1\}$$

 $\log^{(i)} n$ and $\log^{i} n$ are two different functions.

School of UST





例子: 对数迭代

The iterated logarithm is a *very* slowly growing function:

$$\log^* 2 = 1$$

$$\log^* 4 = 2$$

$$\log^* 4 = 2$$
 $\log^* 16 = 3$

$$\log^* 65536 = 4$$

$$\log^* 2^{65536} = 5$$

Since the number of atoms in the observable universe is estimated to be about 10⁸⁰, which is much less than 2^{65536} , we rarely encounter an input size n such $\log^* n \ge 5$ that



斐波那契数: Fibonacci numbers

The Smallest 12 Fibonacci numbers are:

$$F_i = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = 1 \\ F_{i-1} + F_{i-2} & \text{if } i \ge 2 \end{cases}$$





斐波那契数: Fibonacci numbers

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

$$\phi = \frac{1 + \sqrt{3}}{2} = 1$$

$$F_{i} = \frac{\phi^{i} - \hat{\phi}^{i}}{\sqrt{5}} \qquad \text{ \sharp $\stackrel{}{=}$ } \frac{\phi = \frac{1 + \sqrt{5}}{2} = 1.61803\cdots}{\hat{\phi} = \frac{1 - \sqrt{5}}{2} = -0.61803\cdots}$$

由于
$$\left| \hat{\phi} \right| < 1$$
 且 $\left| \frac{\hat{\phi}^i}{\sqrt{5}} \right| < \frac{1}{\sqrt{5}} < 1/2$

$$F_i = \left| \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \right|$$





Homework 3.2:

■ Page 35, 3.2-2, 3.2-4





4 递归: Recurrences

- 在我们分析算法时,通常是将运行时间和空间复杂度表示成输入规模的函数,并且得到有关函数的递归关系式,通过求解递归方程得到运行时间和空间复杂度。
- 例如: 归并排序算法的运行时间可以表示成如下的关系式:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

■ 该递归方程的解为:

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





处理技巧: Technicality

- 在求解递归方程时,通常会忽略一些技术细节,比如,假 定函数自变量是整数,忽视上、下取整符号.
- 通常会忽略递归方程的边界条件,对于足够小的n,将T(n)看作是一个常数,令 $T(n) = \Theta(1)$ 。
- 在有些特殊情况,技术细节非常重要,我们仍然会重视上、 下取整符号和边界条件,以期得到更好的结论。



- 替代法主要包含两个步骤:
- 1) Guess the form of the solution.
- 2) Use mathematical induction to find the constants and show that the solution works.
- 替代法主要用于解的形式易于猜得的情况。
- 替代法既可以用于确定递归方程解的上界,也可以用于确定递归方程解的下界。





Example 4.1:



■ 确定下述递归方程解的上界:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

注意:在证明过程中, 归纳基础不是从 n=1 开始,而是从 n=2 开始.

• 解:猜测其解为 $T(n) = O(n \log_n n)$

我们需要证明对适当选取的常数 c > 0,一定有 $T(n) \le cn \log n$ 成立。可以用数学归纳法加以证明,选取 c > T(2) + 1

另一种更宜接受的方法 是选取 C > T(1)+1,证明 $T(n) \le C n \log n + T(1)$ 则归纳基础可以从 n=1 开始。





如何获得好的猜测解

- 熟能生巧,猜测解通常需要一些经验。
- 可以通过构建递归树获得猜测解。
- 如果某个递归方程与以往的某方程相似,则有理由 猜测其解也相似。
- 没有通用的有效方法获得递归方程正确的猜测解。





NIVERSITY OFSCIENCE & TECHNOLOG OF CHINA

Example 4.2:

■ 求下述递归方程解的上界:

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$$

- 如果没有右端的17,我们已经知道其解为 $T(n) = O(n \log n)$
- 现在仍然猜测其解与上式相同,证明对所有的 n≥1,均有下式成立:

$$T(n) \le c n \log n + c$$







- Sometimes, you make a correct guess of asymptotic bound on the solution of a recurrence, but the math doesn't seem to work out in the induction.
- Usually, the problem is that the inductive assumption isn't strong enough to prove the detailed bound.
- When you face such a situation, revising the guess by subtracting a lower-order term often permits the math to go through.





NIVERSITY OF SCIENCE & TECHNOLOGY OF CHINA

Example 4.3:

■ 求下述递归方程解的上界:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

- 猜测其解为 T(n) = O(n)
- 直接替代,这时发现归纳推理无法继续下去

$$T(n) = c \lfloor n/2 \rfloor + c \lceil n/2 \rceil + 1 = cn + 1$$





Example 4.3(续):

由于无法用直接替代的方法证明,试图将猜测解 无理由的放大,这也是一种常见的错误:

$$T(n) = O(n^2)$$

为了证明这一结论,我们修改原有的结论,将其加强,而不是去减弱它:

$$T(n) \le cn - 1$$

在用归纳法证明的过程中,可以通过引入一个更强的假设,从而证明出一个更强的结论。



小心地雷!



- 错误使用渐近记号
- $T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{if iff} \qquad T(n) = O(n)$
- 猜测 $T(n) \le cn$, 推导出下式:

$$T(n) \le 2(c \cdot \lfloor n/2 \rfloor) + n \le cn + n = O(n)$$

Wrong!!!





变量代换: Changing variables

有时通过变量代换可以简化递归方程的求解,考虑下述看 起来比较复杂的递归方程:

$$T(n) = 2T(\left\lfloor \sqrt{n} \right\rfloor) + \log n$$

设
$$m = \log n$$
, 可得 $T(2^m) = 2T(2^{m/2}) + m$ 令 $S(m) = T(2^m)$, 得到新的递归方程: $S(m) = 2S(m/2) + m$

其解为:

$$S(m) = O(m \log m)$$

$$T(n) = T(2^m) = S(m) = O(m\log m) = O(\log n \log \log n)$$





Homework 4.1:



■ Page 40 , 4.1-1, 4.1-3, 4.1-5





4.2 递归树--The recursion-tree

- 替代法可以用来证明某个递归方程的解是正确的,但有时 要得到一个准确的猜测解是困难的。画一个递归树可以有 助于获得一个好的猜测解。
- 在递归树中,每个结点代表一个相应子问题的代价;
- 递归树中每行结点的代价之和(每行代价)称为行和;
- 对所有的行和求和,可得总的代价。



递归树用途:

- Recursion trees are particularly useful when the recurrence describes the running time of a divide-and-conquer algorithm.
- 由于递归树方法通常用来获取好的猜测解,其解的正确 性可以用替代法来证实。因此,<mark>在构造递归树时通常省</mark> 略一些无关紧要的细节,如:上取整和下取整等。
- 如果在构造递归树以及求和时就非常认真仔细,也可以 直接用递归树方法求解递归方程。





NIVERSITY OFSCIENCE & TECHNOLOGY OF CHINA

Example 4.4:

■ 求解递归方程:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

解:首先求该方程解的一个上界,由于一般情况上整和下整函数对递归方程解的影响不大,将原方程简化为:

$$T(n) = 3T(n/4) + cn^2$$

其中 C 是一个常数。

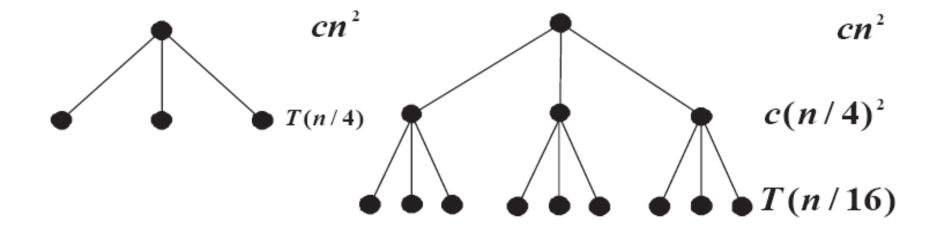






Example 4.4 (续):

■ 为了叙述时的方便,假定*n*为4的方幂。下图给出了方程所对应的递归树的前三层。



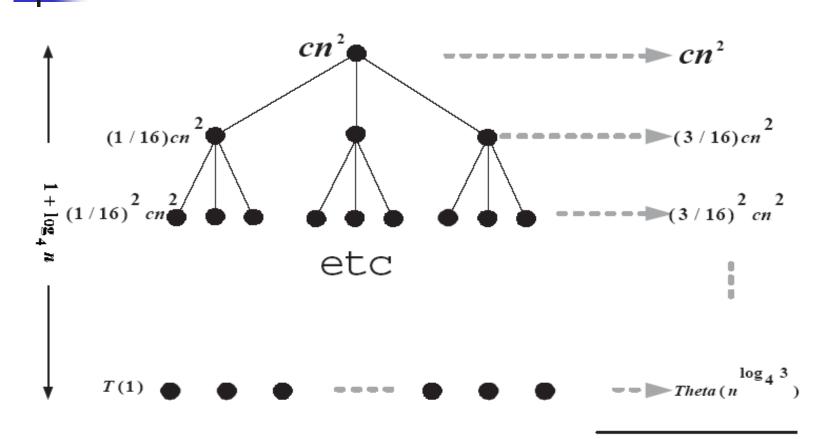
■ 对该递归树继续展开,当子问题规模为1时终止,最终可得到完整的递归树





Example 4.4 (续):





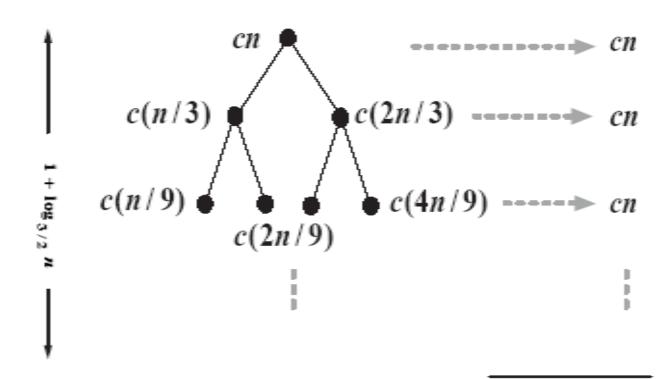
Total: $O(n^2)$





Example 4.5:

• 求解递归方程: T(n) = T(n/3) + T(2n/3) + O(n)



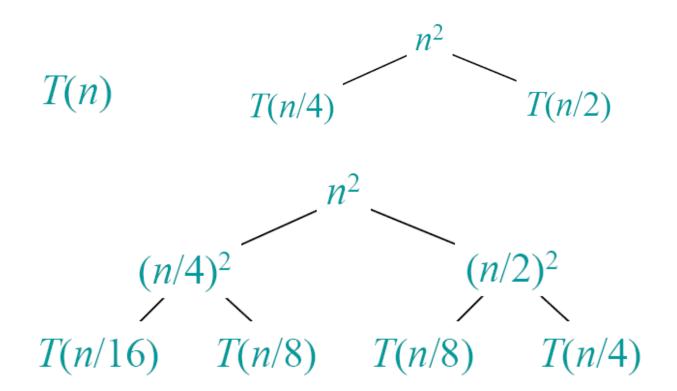
Total: $O(n \log n)$





Example 4.6:

■ 求解递归方程 $T(n) = T(n/4) + T(n/2) + n^2$

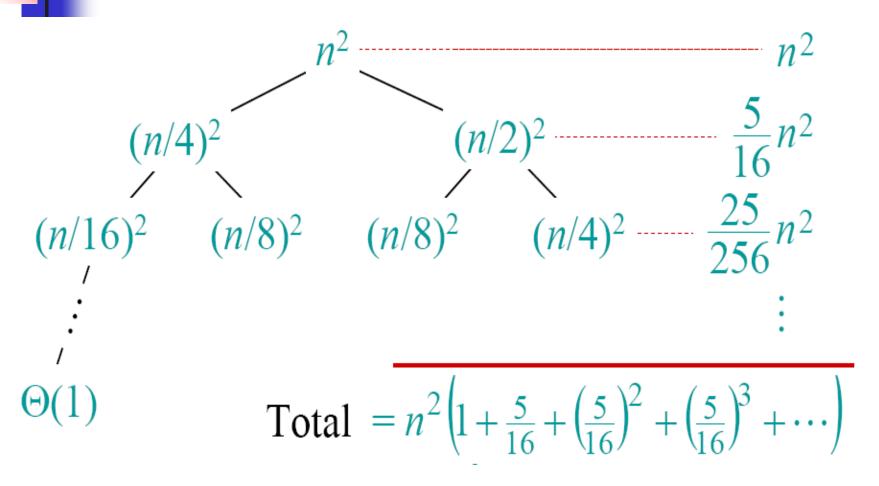






NIVERSITY OF SCIENCE & TECHNOLOGY OF CHINA

Example 4.6 (续):







小结



■ 如何求 Row sum 与 Total sum:

在用递归树方法时,关键是<mark>求出行和与总的代</mark>价,为了求得行和,一个有效的办法是根据递归方程本身,推导出下一行的和与上一行的和之间的关系,由此,可使求和过程变得有章可循,从而也更为简单。







Homework 4.2:

■ Page 43 , 4.2-2, 4.2-5

School of CST



■ 主方法主要适用于下列形式的递归方程:

$$T(n) = aT(n/b) + f(n) \tag{4.5}$$

- 其中 a ≥ 1, b > 1, 且 f 是渐近非负的。
- 用 n/b 代替 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$ 。
- Then T(n) can be bounded asymptotically as Master theorem .



主定理: The master theorem

- If $f(n) = O(n^{\log_b a \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$ o
- If $f(n) = \Theta(n^{\log_b a} \log^k n)$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant c < 1 and all sufficiently large n, then $T(n) = \Theta(f(n))$





NIVERSITY OF SCIENCE & TECHNOLOGY OF CHINA

注意事项

- It is important to realize that the three cases do not cover all the possibilities for f(n).
- There is a gap between cases 1 and 2 when f(n) is smaller than but not polynomially smaller.
- Similarly, there is a gap between cases 2 and 3 when f(n) is larger than but not polynomially larger.
- If the function f(n) falls into one of these gaps, or if the regularity condition in case 3 fails to hold, the master method cannot be used to solve the recurrence.





Example 4.7: Not polynomially larger

$$T(n) = 3T(n/3) + n \cdot \log^2 n$$

東中: a = b = 3, $f(n) = n \log^2 n$, f(n) is asymptotically larger than $n^{\log_b a} = n$, but $f(n)/n^{\log_b a} = \log^2 n$ is less than n^{ε} , for any positive $\varepsilon > 0$, master method does not apply for this recurrence.





Example 4.8: Using the master method

- T(n) = 9T(n/3) + n
- 解: a = 9, b = 3, f(n) = n, $n^{\log_b a} = n^2$, 取 $\varepsilon = 0.5$, 则 $f(n) = O(n^{\log_b a \varepsilon})$,因此,由主定理的case 1 可知其解为:

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$



Example 4.9: Using the master method

- T(n) = T(2n/3) + 1
- \mathbf{m} : $a = 1, b = 3/2, f(n) = 1, \quad n^{\log_b a} = 1$, \mathbf{M}

$$f(n) = \Theta(n^{\log_b a})$$
 ,由主定理的case 2 知其解为

$$T(n) = \Theta(n^{\log_b a} \cdot \log n) = \Theta(\log n)$$

School of CST



- $T(n) = 3T(n/4) + n \log n$
- **解**: a = 3, b = 4, $f(n) = n \log n$, $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$ 取 $\varepsilon = 0.1$, 则 $f(n) = \Omega(n^{\log_b a + \varepsilon})$,

 而且 $af(n/b) = 3f(n/4) = 3n/4 \cdot \log(n/4)$ $\leq 3/4 \cdot n \log n = cf(n)$

其中 c = 3/4, 由定理的case 3 知方程的解为 $T(n) = \Theta(f(n)) = \Theta(n \log n)$





Example 4.11: Using the master method

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

解: 主方法对该递归方程不适用,因为 a = 2, b = 2, $n^{\log_b a} = n$,该递归方程恰好处于主方法的 case 2 和 case 3 之间.

 $T(n) = 4T(n/2) + n^2/\log n$

解:主方法同样对该递归方程不适用。因为,a = 4,b = 2, $f(n) = n^2/\log n$,该递归方程恰好处于主方法的 case 1 和 case 2 之间.





School of

Homework 4.3:

■ Page 45 , 4.3-2, 4.3-3