数据结构与算法

第二章 数学基础

裴文杰

计算机科学与技术学院 教授

本章内容

阅读《算法导论》第三章

如何描述算法的效率

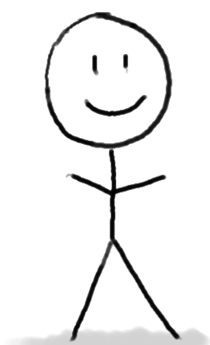
- 记录算法实现的程序在机器上实际运行的时间？这样会有什么问题呢？
  - 实现代码的语言的效率差别很大
  - 代码的优化程度
  - 机器的运算速度，指令集
  - 。。。

对比不同算法的实际运行时间非常困难

- 我们能不能找出一种方法，排除这些干扰因素，只关注算法本身的效率？

增长的阶

- 核心思想
  - 关注算法的运行时间是如何随着问题的规模（也即输入大小，输入规模）而变化的



Algorithm designer

Here is my algorithm!

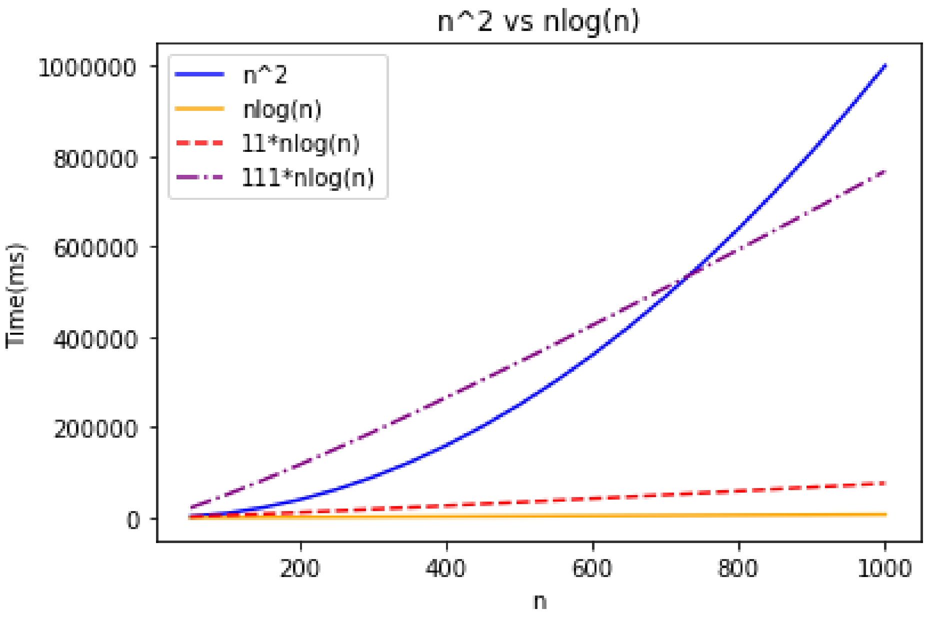
```
Algorithm:
  Do the thing
  Do the stuff
  Return the answer
```

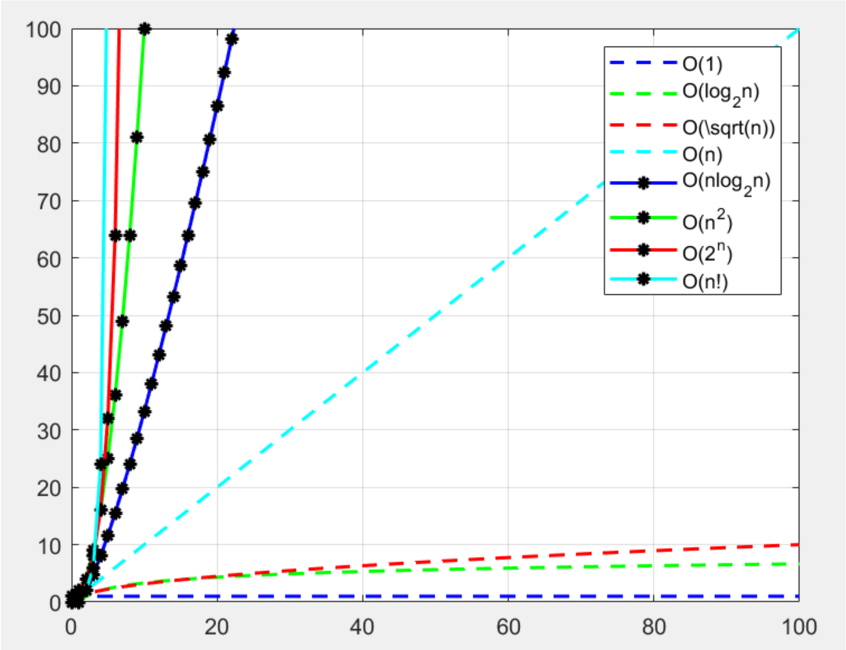
HERE IS AN INPUT!

增长的阶

- 算法增长的阶也即增长率，增长量级或时间复杂度（time complexity）
- 只与输入问题的规模相关
- 一个算法比另一个算法“效率高”：如果它的运算时间（原子操作步数）随着输入规模的增长比另一个算法增长的慢
- 只关注增长最快的项（忽略低阶项，保留最高阶项）
- 忽略常系数

增长的阶





1. 影响增长率的关键是函数的（最高）阶
2. 函数的系数大小随着n的增大不是很关键

增长的阶

Quick log refresher

All logarithms in this course are base 2

log(n) : how many times do you need to divide n by 2 in order to get down to 1?

32	64
16	32
8	16
4	8
2	4
1	2
	1
log(32) = 5	log(64) = 6

log(128) = 7
log(256) = 8
log(512) = 9
.
.
.
log(**number of particles in the universe**) < 280

log(n) grows very slowly with n.

增长函数

- **渐进效率**
  - 输入规模无限增加时，在极限中，算法的运行时间如何随着输入规模的变大而增加。
  - 忽略低阶项和最高阶的系数
  - 只考虑最高阶(增长的阶)
- **典型的增长阶**
  - $\Theta(1)$, $\Theta(\lg n)$, $\Theta(\sqrt{n})$,$\Theta(n)$, $\Theta(n\log n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(2^n)$, $\Theta(n!)$
- 增长（渐进）记号: O, Θ, Ω, o, ω.
  - 既可以用来刻画算法运行时间效率，也可以描述空间效率

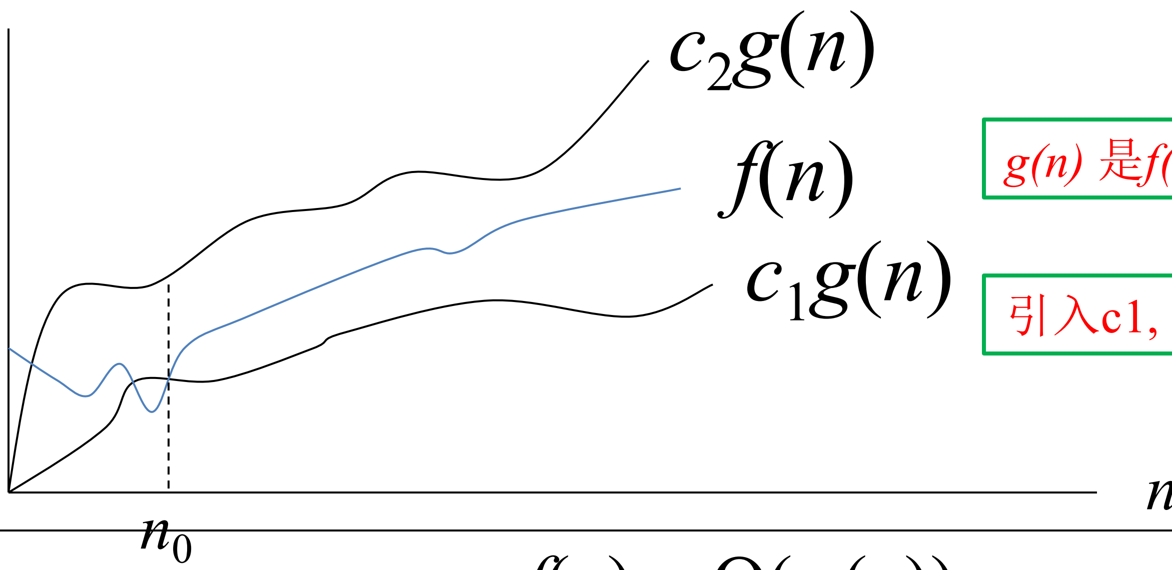
同阶函数集合

定义：

$\Theta(g(n))=\{f(n) \mid \exists c_1, c_2>0, \; n_0, \; \forall n \geq n_0, \; 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

称为与$g(n)$同阶的函数集合。我们称为$g(n)$是$f(n)$的渐进紧界。

- 如果$f(n)\in\Theta(g(n))$，$g(n)$与$f(n)$同阶
- $f(n)\in\Theta(g(n))$，记作$f(n)=\Theta(g(n))$



$g(n)$ 是$f(n)$的渐进紧界

引入c1, c2 排除了最高阶系数的影响

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

$\Theta(g(n))$ 函数的例子

- 证明 $1/2n^2 - 3n = \Theta(n^2)$.
  - 需要证明： $\exists c_1, c_2>0,\ n_0,\ \forall n \geq n_0,\ c_1 n^2 \leq 1/2n^2 - 3n \leq c_2 n^2$
  - 即： $c_1 \leq 1/2 - 3/n \leq c_2$
  - 对于任意$n \geq 1$, $c_2 \geq ½$; 且对于任意$n \geq 7$， $c_1 \leq 1/14$
  - 因此 $c_1 = 1/14$, $c_2 = ½$, $n_0 = 7$.

当然也可以选择其他合适的n_0和c_1, c_2

- 证明 $6n^3 \neq \Theta(n^2)$. （反证法）
  - 如果存在c_1、$c_2>0$， n_0使得当$n \geq n_0$时， $c_1 n^2 \leq 6n^3 \leq c_2 n^2$ 。
  - 也即$c_1/6 \leq n \leq c_2/6$, 显然与 $\forall n>n_0,\ c_1 g(n) \leq f(n) \leq c_2 g(n)$ 矛盾

$\Theta(g(n))$ 函数的例子（重要）

- 通常$f(n)=an^2+bn+c = \Theta(n^2)$, 其中$a,b,c$是常数且 $a>0$.
- $p(n) = \sum_{i=0}^{d} a_i n^i$,其中$a_i$是常数且$a_d>0$, $p(n) = \Theta(n^d)$.
- $\Theta(n^0)$或者$\Theta(1)$，常数时间复杂性.

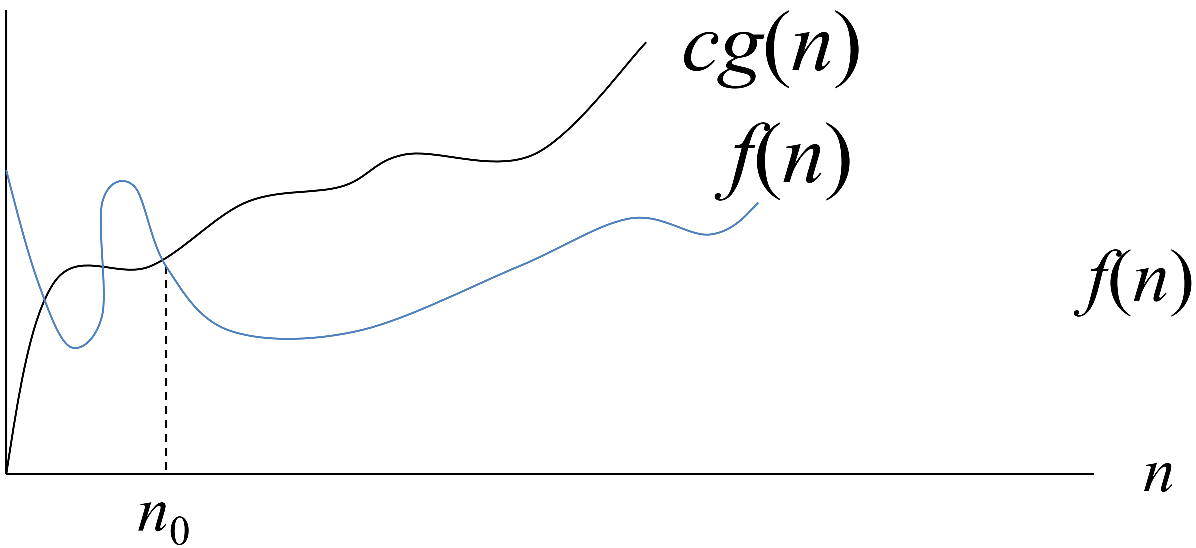
低阶项和最高阶项的系数都可以省略

低阶函数集合

定义：

对于给定的函数$g(n)$,$O(g(n))=\{f(n)$: 存在正常数c和n_0 满足对于所有$n \geq n_0$, $0 \leq f(n) \leq cg(n)\}$，我们称为$g(n)$是$f(n)$的渐进上界（upper bound）

- 记作$f(n) \in O(g(n))$, 或简记为$f(n) = O(g(n))$.



$$f(n) = O(g(n))$$

$\Theta(g(n))$和$O(g(n))$的关系

- $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$，反过来不一定成立
- Θ标记强于O标记.
- $\Theta(g(n)) \subseteq O(g(n))$
  - $an^2+bn+c = \Theta(n^2)$, 且$=O(n^2)$
  - $an+b = O(n^2)$. 为什么？
  - $n = O(n^2)$!!!
- O标记, 表示渐进上界
- Θ标记, 表示渐进紧界
- 一些讨论：**如果一种算法的最好运行时间是$O(n)$，最坏是$O(n^2)$，那么：**
  - $O(n^2)$ 适用于所有的输入，即使对于最好情况也成立，因为$O(n) \in O(n^2)$.
  - 然而$\Theta(n^2)$只能用于最坏情况，不能应用到最好情况，因为$\Theta(n) \neq \Theta(n^2)$.

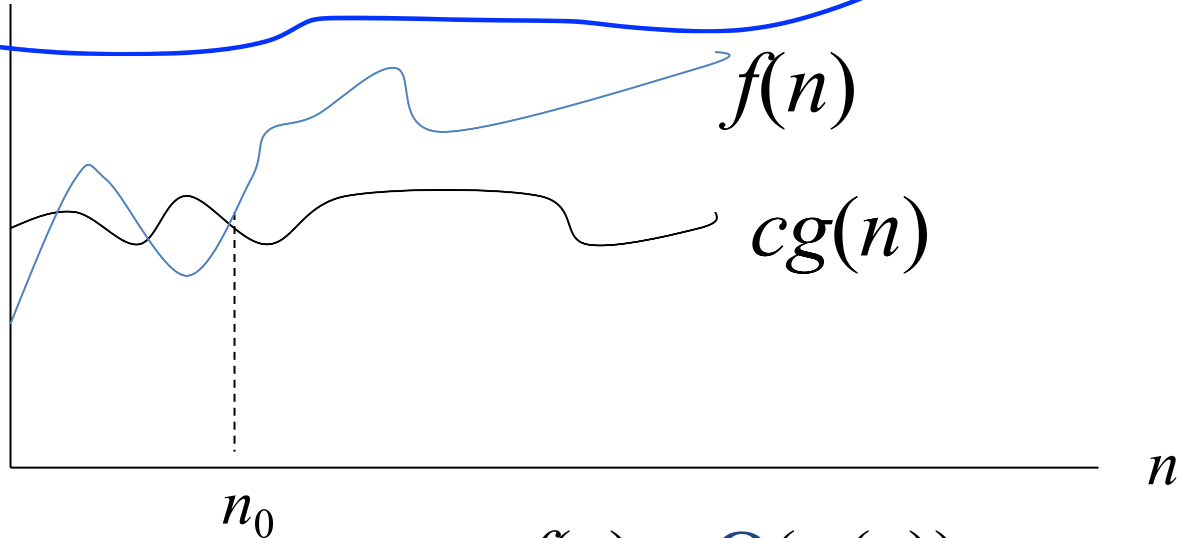
如果$f(n)=O(n^k)$, k是常数，则称$f(n)$是多项式界限的。

高阶函数集合

定义：

对于给定的函数$g(n)$，$\Omega(g(n))$=$\{f(n)$：存在正常数c和n_0，使得对于所有$n \geq n_0$，$0 \leq cg(n) \leq f(n)\}$

- 记作$f(n) \in \Omega(g(n))$, 或简记为$f(n) = \Omega(g(n))$.
- 我们称为$g(n)$是$f(n)$的渐进下界（lower bound）



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

O, Θ, Ω标记的关系

- 对于$f(n)$和$g(n)$, $f(n) = \Theta(g(n))$当且仅当$f(n) = O(g(n))$且$f(n) = \Omega(g(n))$.
- O: 渐进上界
- Θ: 渐进紧（确）界
- Ω: 渐进下界

关于Ω 标记

- 可以用来描述运行时间的最好情况
- 对所有输入都正确
  - **如果一种算法的最好运行时间是$\Omega(n)$，最坏运行时间是$\Omega(n^2)$，但是这种情况说运行时间是$\Omega(n^2)$则有误，因为直接说运行时间，不加修饰语，意味着需要对所有可能的输入都成立。**
- 可以用来描述问题
  - 排序问题的时间复杂性是$\Omega(n)$

问题的下界是解决该问题的所有算法中所需要的最小时间复杂性。

等式和不等式中的渐进符号

- 当渐进符号单独出现在等式或不等式中时， 例 如 $f(n)=O(n^2)$, 此时意指集合的成员关系 $f(n) \in O(n^2)$。

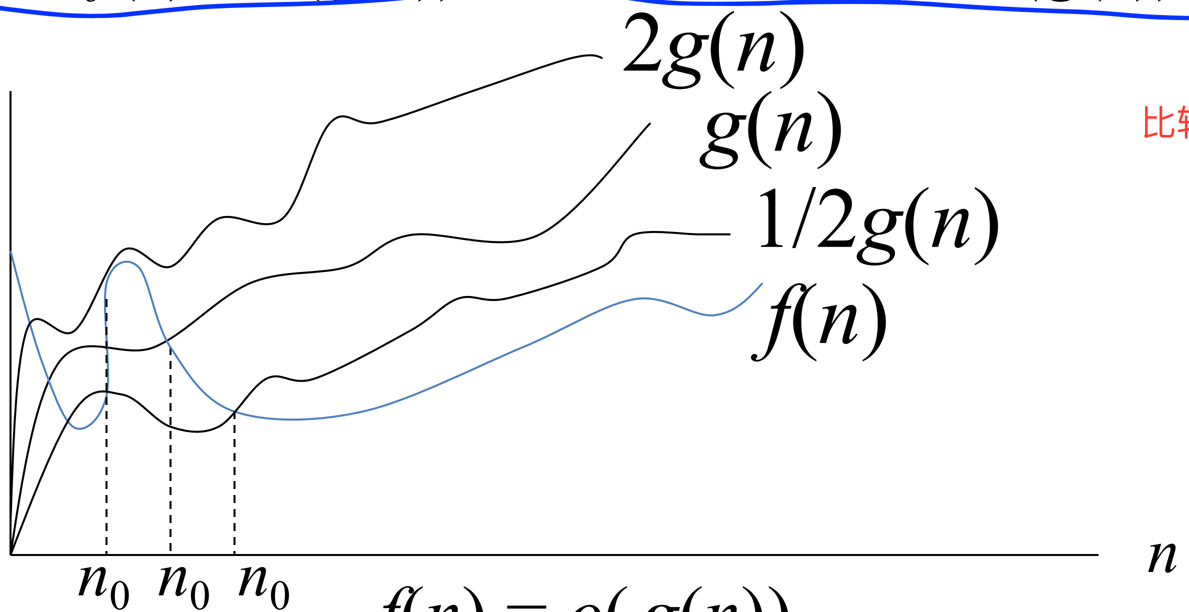
- 当渐进符号出现在某个公式中时，例如 $2n^2+3n+1 = 2n^2+ \Theta(n)$，此时意指$2n^2+3n+1=2n^2+f(n)$, 其中$f(n)$是集合$\Theta(n)$的某个函数，比如$f(n) = 3n+1$

严格低阶函数集合

定义：

给定一个函数$g(n)$,$o(g(n)) = \{f(n)$: 对于**任意**正常数c，存在一个正数n_0，从而对所有$n \geq n_0$，满足$0 \leq f(n) < cg(n)\}$

➢ 记作$f(n) \in o(g(n))$, 或者简写为 $f(n) = o(g(n))$.



$f(n) = o(g(n))$

比较松

关于o标记

- O标记可能是或不是紧的
  - $2n^2 = O(n^2)$ 是紧的, 但$2n = O(n^2)$不是紧的.
- o标记用于标记上界但不是紧的情况
  - $2n = o(n^2)$, 但是$2n^2 \neq o(n^2)$.
- 区别: 存在某个正常数c在O标记中, 但所有正常数c在o标记中.

$$f(n) = o(g(n)) \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

证. 由于 $f(n)=o(g(n))$, 对任意$\varepsilon>0$, 存在n_0, 当$n \geq n_0$时,

$$0 \leq f(n) < \varepsilon g(n),$$

即 $0 \leq \dfrac{f(n)}{g(n)} \leq \varepsilon$. 于是, $\lim\limits_{n \to \infty} \dfrac{f(n)}{g(n)} = 0$.

严格低阶函数

- **例 1：** 证明 $2n = o(n^2)$

证明：对∀c>0，欲 $2n < cn^2$，必 $2<cn$，即 $\frac{2}{c} < n$。所以，对∀c>0，

当 $n_0 = \frac{2}{c}+1$ 时，对所有 $n \geq n_0$，都有 $2n<cn^2$

注：可以对不同的c选取不同的 n_0

- **例 2：** 证明 $2n^2 \neq o(n^2)$

证明：当c=1时，对于任何 n_0，当 $n \geq n_0$，$2n^2<cn^2$ 都不成立

严格高阶函数集合

- 对于给定函数$g(n)$,
  - $\omega(g(n))=\{f(n)$: 对于**任意**正常数c, 存在正数n_0 对于$n \geq n_0$， $0 \leq cg(n) < f(n)\}$
  - 记作$f(n) \in \omega(g(n))$, 或者简记为$f(n) = \omega(g(n))$.
- ω标记与Ω标记的关系，类似o标记与O标记的关系，ω表示不紧的下界.
  - $n^2/2 = \omega(n)$, 但$n^2/2 \neq \omega(n^2)$
- $f(n) = \omega(g(n))$当且仅当$g(n)=o(f(n))$.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

渐进符号的性质

- 传递性: 所有五个标记
  - $f(n) = \Theta(g(n))$ 且$g(n) = \Theta(h(n))$ ➔ $f(n) = \Theta(h(n))$
- 自反性: O, Θ, Ω .
  - $f(n) = \Theta(f(n))$
- 对称性: Θ
  - $f(n) = \Theta(g(n))$ 当且仅当$g(n) = \Theta(f(n))$
- 反（转置）对称性:
  - $f(n) = O(g(n))$当且仅当 $g(n) = \Omega(f(n))$.
  - $f(n) = o(g(n))$当且仅当 $g(n) = \omega(f(n))$.

不同的增长记号对比

同阶函数集合：$\Theta(g(n))=\{f(n) \mid \exists c_1, c_2>0,\ n_0,;\ \ \forall n>n_0,$
$c_1g(n)\leq f(n)\leq c_2g(n)\}$

低阶函数集合：$O(g(n))=\{f(n) \mid \exists c，\ n_0>0;\ \ \forall n\geq n_0;$
$0\leq f(n) \leq cg(n)\}$

严格低阶函数集合：$o(g(n))=\{f(n) \mid \forall c>0；\ \ \exists n_0>0，$
$\forall n\geq n_0;\ \ 0\leq f(n) < cg(n)\}$

高阶函数集合：$\Omega(g(n))=\{f(n): \exists c，\ n_0>0；\ \ \forall n\geq n_0;$
$0 \leq cg(n) \leq f(n)\}$

严格高阶函数集合：$\omega(g(n))=\{f(n): \forall c>0；\ \ \exists n_0>0，$
$\forall n\geq n_0;\ \ 0 \leq cg(n) < f(n)\}$

注意

!

并非所有函数都是渐进可比的，即对于函数 $f(n)$ 和 $g(n)$，可能 $f(n) \neq O(g(n))$ 且 $f(n) \neq \Omega(g(n))$，例如 n 和 $n^{1+sin(n)}$

$0 \le 1+sin(n) \le 2$, 且 $sin(n)$ 是周期函数

本讲内容

2.1　计算复杂性函数的阶

2.2　和式的估计与界限

2.3　递归方程

为什么需要和式的估计与界限

for $l=2$ to n

  for $i=1$ to $n-l+1$ do

    $j=i+l-1;$

    $m[i, j]=\infty;$

    for $k=i$ to $j-1$ do

      $q=m[i, k]+m[k+1, j]+p_{i-1}p_kp_j$

      if $q<m[i, j]$ then $m[i,j]=q;$

当一个算法包含循环体时，运行时间需要求和。

阅读《算法导论》附录A

和式的估计

1. 线性和

$$\sum_{k=1}^{n}(ca_k + b_k) = c\sum_{k=1}^{n}a_k + \sum_{k=1}^{n}b_k$$

和式的估计

2 级数

- 等差级数 $\sum_{k=0}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$

- 几何级数

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1}-1}{x-1} \quad (x \neq 1)$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad (|x| < 1)$$

- 调和级数

$$\sum_{k=1}^{n} \frac{1}{\mathrm{k}} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \ln n + O(1)$$

怎么求？请看后续分解。

和式的估计

2 级数

- 裂项级数

$$\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$$

$$\sum_{k=0}^{n-1}(a_k - a_{k+1}) = a_0 - a_n$$

$$\sum_{k=1}^{n-1}\frac{1}{k(k+1)} = \sum_{k=1}^{n-1}\left(\frac{1}{k} - \frac{1}{k+1}\right) = 1 - \frac{1}{n}$$

$$\lg\left(\prod_{k=1}^{n} a_k\right) = \sum_{k=1}^{n}\lg a_k$$

3. 计算和的界限

➢ 采用数学归纳法

使用相同的C值

例1. 证明$\sum_{k=0}^{n} 3^k = O(3^n)$

注意：使用数学归纳法时，对于不同的n，使用相同的c值。

证明：

为什么是3/2？

对于$c \geq \frac{3}{2}$，存在n_0，当$n > n_0$时，$\sum_{k=0}^{n} 3^k \leq c3^n$

当$n = 0$时，$\sum_{k=0}^{n} 3^k = 1 \leq c = c3^n$

设$n \leq m$时成立，令$n = m + 1$，则

第二数学归纳法

$$\sum_{k=0}^{m+1} 3^k = \sum_{k=0}^{m} 3^k + 3^{m+1} \leq c3^m + 3^{m+1} = c3^{m+1}\left(\frac{1}{3} + \frac{1}{c}\right) \leq c3^{m+1}$$

和式的估计

3. 计算和的界限

- 采用数学归纳法

对于给定的函数$g(n)$，

$O(g(n)) = \{f(n) | \exists c > 0和n_0, 对于\forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$

例2. 错误证明$\sum_{k=1}^{n} k = O(n)$。

证明：

当$n = 1$时，$\sum_{k=1}^{n} k = 1 \leq c1 = O(1)$只需$c$ 大于等于1

设$n \leq m$时成立，令$n = m + 1$，则

使用了不同的C

错误

$$\sum_{k=1}^{m+1} k = \sum_{k=1}^{m} k + (m+1) \leq cm + (m+1) = (c+1)m + 1 = O(m)$$

要证明的应该是上式$\leq c(m+1)$，即$cm + (m+1) \leq c(m+1)$ 则 $c \geq m+1$才满足条件。

错在$O(n)$的常数c随n的增长而变化，不是常数。

要证明$\sum_{k=1}^{n} k = O(n)$，需证明：对某个确定值的$c > 0$，$\sum_{k=1}^{n} k \leq cn$。

和式的估计

3. 计算和的界限

- **直接求解**，通过用级数中最大项的界作为其他项的界

例1. $\sum_{k=1}^{n} k \le \sum_{k=1}^{n} n = n^2$。

例2. $\sum_{k=1}^{n} a_k \le n \times max\{a_k\}$。

和式的估计

3. 计算和的界限

➢ **直接求解**，通过求解每一项的界

当一个级数以几何级数（指数级数）增长时，用最大项作为每一项的上界过于宽松，并不理想。

例3. 设对于所有$k \geq 0, a_0 \geq 0, 0 \leq \frac{a_{k+1}}{a_k} \leq r < 1$，求$\sum_{k=0}^{n} a_k$的上界。

解：

$\frac{a_1}{a_0} \leq r \Rightarrow a_1 \leq a_0 r$

$\frac{a_2}{a_1} \leq r \Rightarrow a_2 \leq a_1 r \leq a_0 r^2$

$\frac{a_3}{a_2} \leq r \Rightarrow a_3 \leq a_2 r \leq a_1 r^2 \leq a_0 r^3$

$\cdots$

$\frac{a_k}{a_{k-1}} \leq r \Rightarrow a_k \leq a_{k-1} r \leq \cdots \leq a_1 r^{k-1} \leq a_0 r^k$

于是，$\sum_{k=0}^{n} a_k \leq \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = a_0 \frac{1}{1-r} \quad (|r| < 1)$。

和式的估计

3. 计算和的界限

- **直接求解**，通过求解每一项的界

例4. 求$\sum_{k=1}^{\infty}\left(\frac{k}{3^k}\right)$的上界。

解：使用例3的方法

$$\frac{\frac{k+1}{3^{k+1}}}{\frac{k}{3^k}}=\frac{1}{3}\times\frac{k+1}{k}=\frac{1}{3}\times\left(1+\frac{1}{k}\right)\leq\frac{2}{3}=r<1,$$

于是：$\sum_{k=1}^{\infty}\frac{k}{3^k}\leq\sum_{k=0}^{\infty}a_1r^k=\sum_{k=0}^{\infty}\frac{1}{3}\times\left(\frac{2}{3}\right)^k=\frac{1}{3}\times\left(\frac{1}{1-\frac{2}{3}}\right)=1$

和式的估计

3. 计算和的界限

- **直接求解**，分割（分裂）求和
  - **将和式**项分割，并忽略其常数个起始项，对常数k_0:

$$\sum_{k=1}^{n} a_k = \sum_{k=1}^{k_0-1} a_k + \sum_{k=k_0}^{n} a_k \geq \Theta(1) + \sum_{k=k_0}^{n} a_k$$

例 5. 用分割求和的方法求$\sum_{k=1}^{n} k$的下界

$$\sum_{k=1}^{n} k = \sum_{k=1}^{n/2} k + \sum_{k=n/2}^{n} k \geq \sum_{k=1}^{n/2} 0 + \sum_{k=n/2}^{n} \frac{n}{2} = \left(\frac{n}{2}\right)^2 = \Omega(n^2)$$

和式的估计

3. 计算和的界限

- **直接求解**，分割（分裂）求和
  - **将和式**项分割，并忽略其常数个起始项，对常数k_0:

例6. 求 $\sum_{k=0}^{\infty}\frac{k^2}{2^k}$ 的上界。

解：当$k \geq 3$时，

$$\frac{(k+1)^2/2^{k+1}}{k^2/2^k}=\frac{(k+1)^2}{2k^2}\leq\frac{8}{9}$$

于是，$\sum_{k=0}^{\infty}\frac{k^2}{2^k}=\sum_{k=0}^{2}\frac{k^2}{2^k}+\sum_{k=3}^{\infty}\frac{k^2}{2^k}\leq 0+\frac{1}{2}+1+\sum_{k=3}^{\infty}\frac{9}{8}\left(\frac{8}{9}\right)^{k-3}$

$$\leq\frac{3}{2}+\sum_{k=0}^{\infty}\frac{9}{8}\left(\frac{8}{9}\right)^{k}=\frac{3}{2}+\frac{9}{8}*\frac{1}{1-\frac{8}{9}}=\frac{93}{8}=O(1)$$

为什么要从3分割开？
直接使用之前例3的方法为什么不行？

和式的估计

3. 计算和的界限

- **直接求解，分割（分裂）求和**

例7. 求调和级数$H_n = \sum_{k=1}^{n} \frac{1}{k}$ 的上界。

调和级数的解法一

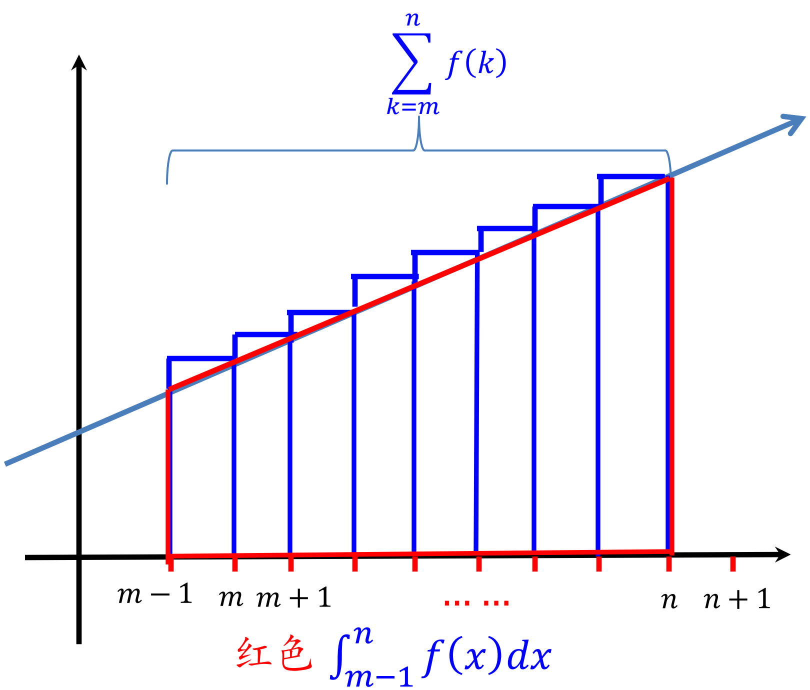
解：$\sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}\right)$

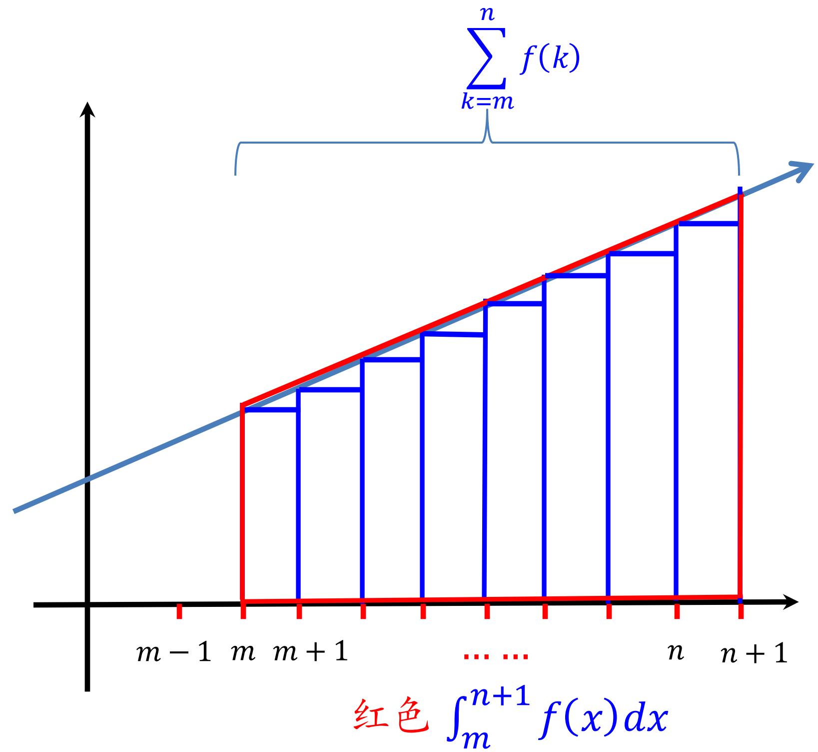
$$+ \left(\frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \frac{1}{11} + \frac{1}{12} + \frac{1}{13} + \frac{1}{14} + \frac{1}{15}\right) + \ldots\ldots$$

$$\le \sum_{i=0}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \le \sum_{i=0}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} = \sum_{i=0}^{\lfloor \log n \rfloor} 1 \le \log n + 1$$

积分求和的近似

如果f(k)单调递增，则 $\int_{m-1}^{n} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)dx$





$\sum_{k=m}^{n} f(k) = \sum_{k=m}^{n} f(k)\Delta x \ge \int_{m-1}^{n} f(x)dx\,, f(m-1) < f(n), \Delta x = 1$

$\sum_{k=m}^{n} f(k) = \sum_{k=m}^{n} f(k)\Delta x \le \int_{m}^{n+1} f(x)dx$

积分求和的近似

如果f(k)单调递减，则 $\int_{m}^{n+1} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)dx$

例 8： 采用积分求和的方式计算调和级数的上下界

调和级数的解法二

$$\sum_{k=1}^{n}\frac{1}{k} \ge \int_{1}^{n+1}\frac{dx}{x} = \ln(n+1)$$

$$\sum_{k=2}^{n}\frac{1}{k} \le \int_{1}^{n}\frac{dx}{x} = \ln n$$

积分求和，比较紧

$$\ln(n+1) \le \sum_{k=1}^{n}\frac{1}{k} \le \ln n + 1$$

本讲内容

递归方程

- 斐波那契数列

$$F(n) = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ F(n-1) + F(n-2), & n > 2 \end{cases}$$

  - 递归解法

```
public int fib(int n) {
    if (n < 1) {
        return -1;
    }
    if (n == 1 || n == 2) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}
```

递归方程

- **方程** $T(n)=2\,T(\frac{n}{2})+11n$ 表示了一种递归关系。
- 递归方程描述了$T(n)$与$T(<n)$之间的关系
- 挑战：

给定一个表示代价的递归方程$T(n)$，能否找到$T(n)$的解析解（closed-form expression）？

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

递归方程的初始条件

- 递归方程需有基本情况或初始条件。
  - $T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$ with $T(1) = 1$

不同于

  - $T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$ with $T(1) = 1000000000$
- 递归方程描述了$T(n)$与$T(< n)$之间的关系
- 然而，$T(1) = O(1)$，因此，我们可以忽略具体的值

忽略边界条件：
对于一个常量规模的输入，算法运行时间为常量，对于足够小的n，T(n) 为常量，改变T(1)不会改变函数的增长阶。

求解递归方程的三个主要方法

- 替换（代入）方法:
  - 首先猜想（界）；
  - 然后用数学归纳法证明.
- 迭代（递归树）方法:
  - 画出递归树：节点表示不同层次的递归调用产生的代价；
  - 把递归式转化为一个和式；
  - 然后用估计和的方法来求解。
- Master定理（主定理，主方法）方法:
  - 求解型为$T(n)=aT(n/b)+f(n)$的递归方程

阅读《算法导论》4.3-4.6

替换（代入）方法

- 步骤
  - 首先猜想界
  - 然后用数学归纳法

替换（代入）方法

例1：求解 $T(n)=2T(\lfloor n/2 \rfloor)+n, T(1)=1$ 的上界

- 根据经验，猜测其解为 $T(n) = O(n\log n)$，要求证明：$\exists C>0$, $T(n) \le cn\log n$。
- 归纳法证明（第二数学归纳法）：

  1) 假设对于正整数$m<n$都成立，

  那么$T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$；

  2) 将其代入递归式，得到：

$$
\begin{aligned}
T(n) &\le 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor))+n \le c\, n \log(n/2) + n \\
&= cn \log n - cn \log 2 + n \\
&= cn \log n - cn + n \\
&\le cn \log n\text{，当 } c \ge 1
\end{aligned}
$$

注意：使用数学归纳法时，对于不同的n，使用相同的c值。

替换（代入）方法

例1：求解 $T(n)=2T(\lfloor n/2 \rfloor)+n, T(1)=1$ 的上界

该式中，对应的n_0是多少？

- **初始条件不成立时，往后推，看是否成立**
- T(1)<c log1=0，与T(1)=1矛盾
- T(2)= 4<=c 2log2, 只需c>2, 成立

渐进符号只要求：

$\exists\ n_0$，当 $n \geq n_0$时，$T(n) \leq cn\lg n$。

对于大多数递归式而言：

- **扩展边界条件**使得归纳假设对较小的n成立，是一种简单直接的方法。
- 选择足够大的c（调整c的值）有助于处理边界条件

替换（代入）方法

猜测方法I：联想已知的$T(n)$

例2. 求解$T(n)=2T(n/2 + 17) + n$

解：猜测：$T(n)=2T\left(\frac{n}{2}+17\right)+n$ 与 $T(n)=2T\left(\frac{n}{2}\right)+n$ 只相差一个 17.

当 n 充分大时 $T\left(\frac{n}{2}+17\right)$ 与 $T\left(\frac{n}{2}\right)$ 的差别并不大，因为

$\frac{n}{2}+17$ 与 $\frac{n}{2}$ 相差小. 我们可以猜 $T(n)=O(n\lg n)$.

证明：用数学归纳法

替换（代入）方法

猜测方法II：先证明较松的上下界，然后缩小不确定性范围

例3. 求解$T(n) = 2 * T\left(\frac{n}{2}\right) + n$。

解：
首先证明$T(n)= \Omega(n)$, $T(n) = O(n^2)$,
然后逐渐地降低上界，提高下界：
$\Omega(n)$的上一个阶是$\Omega(n\log n)$,
$O(n^2)$的下一个阶是$O(n\log n)$.

替换（代入）方法

细微差别的处理

- 问题：猜测正确，数学归纳法的归纳步似乎证不出来
- 解决方法：
  - 求上界：从猜测中减去一个低阶项，使得归纳假设更强，可能work.
  - 求下界：从猜测中加上一个低阶项，可能就可以了

替换（代入）方法

例 4. 求解 $T(n)=T(\lfloor n/2\rfloor)+T\left(\left\lceil \frac{n}{2}\right\rceil\right)+1$

为什么假设更强能work：
证明时可以得到更强更紧的界，比如本例中两次减去常数b,而证明结论只需要减去1次。

解：（1）我们猜 $T(n)=O(n)$

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

证不出 $T(n)=O(cn)$

（2）减去一个低阶项，猜 $T(n)\le cn-b$，$b\ge 0$ 是常数

证：设当 $\le n-1$ 时成立

$$T(n)=T(\lfloor n/2\rfloor)+T\left(\left\lceil \frac{n}{2}\right\rceil\right)+1\le c\left\lfloor \frac{n}{2}\right\rfloor-b+c\left\lceil \frac{n}{2}\right\rceil-b+1$$

$$=cn-2b+1=cn-b-b+1\le cn-b$$ （只要 $b\ge 1$）。

* c 必须充分大，以满足边界条件。

替换（代入）方法

避免陷阱

例5. 求解 $T(n) = 2 * T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ 。

解：猜测$T(n) = O(n)$

$\boxed{\text{--错！！}}$

证明：用数学归纳法证明$T(n) \le cn$

$$T(n) = 2 * T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \le 2\left(c\left\lfloor \frac{n}{2} \right\rfloor\right) + n \le (c+1)n = O(n)$$

错在哪里：我们需要的证明的是$T(n) \le cn$,
而 从$T(n) \le cn+n$得不到$T(n) \le cn$。

注意：使用数学归纳法时，对于不同的n，使用相同的c值。

替换（代入）方法

变量替换方法:

经变量替换把递归方程变换为熟悉的方程.

例6. 求解$T(n) = 2 * T(\sqrt{n}) + \lg n$

没见过的形式，很难直接猜想边界

解：令$m = \lg n$，则$n = 2^m$，$T(2^m) = 2 * T\left(2^{\frac{m}{2}}\right) + m$

令$S(m) = T(2^m)$，则$T\left(2^{\frac{m}{2}}\right) = S\left(\frac{m}{2}\right)$。

变量替换

于是，$S(m) = 2S\left(\frac{m}{2}\right) + m$。

显然，$S(m) = O(m\lg m)$，即$T(2^m) = O(m\lg m)$。

由于$2^m = n$，$m = \lg n$，$T(n) = O(\lg n \lg(\lg n))$。

迭代（递归树）方法

方法：

- 画出递归树
- 循环地展开递归方程
- 把递归方程转化为和式
- 然后使用求和技术解之

迭代（递归树）方法

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

分治思想：通过解一个问题的子问题，来解一个问题

a ： 子问题数量

b ： 子问题从原问题缩小的比例，n/b为子问题规模

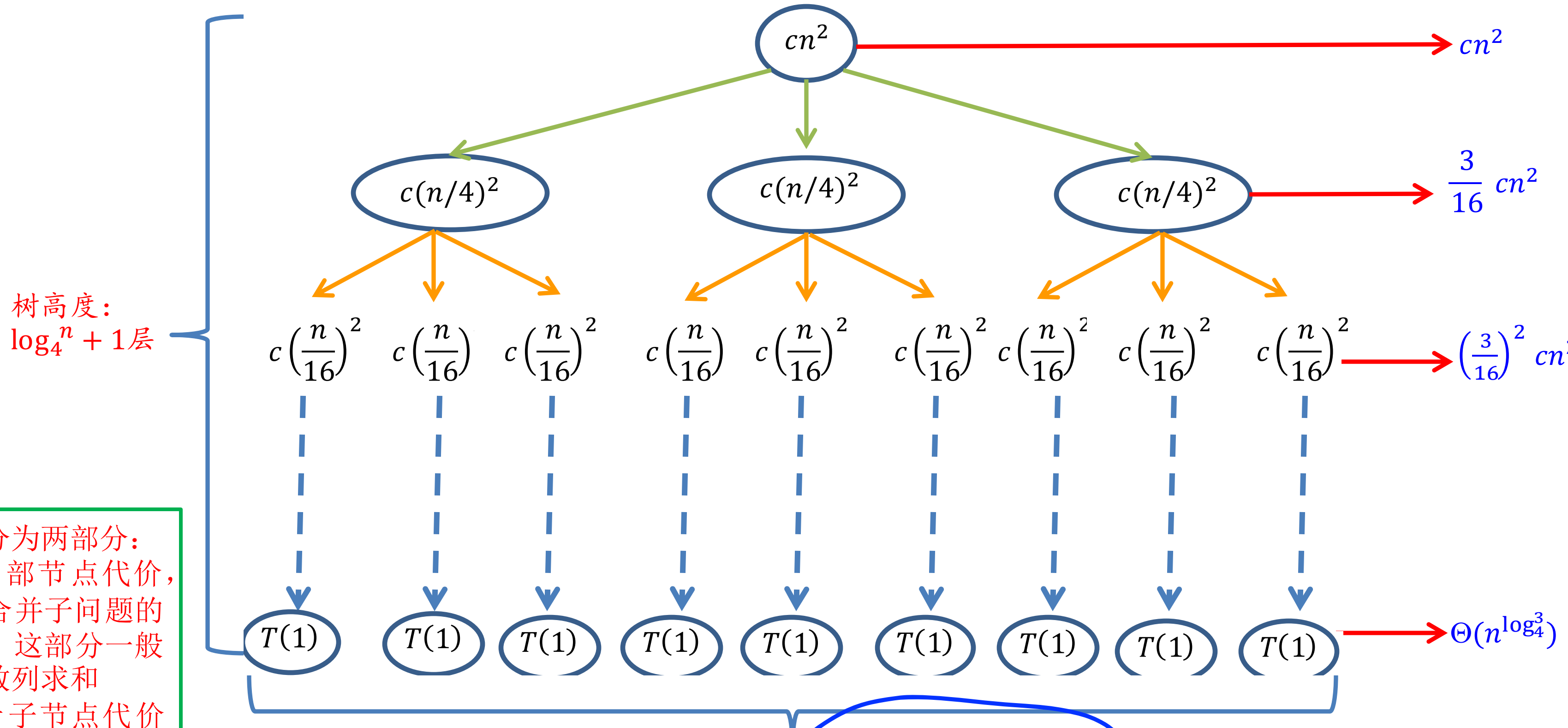
f(n)：将问题分解和子问题解整合的代价

迭代（递归树）方法

- 根结点表示递归调用顶层的代价
- 内部节点，表示不同层次递归调用产生的代价，即（分解和合并）子问题的代价
- 树的分枝数量取决于子问题的数量
- 叶节点表示边界条件值
- 每个节点表示一个单一子问题，节点代价及其其子树所有节点代价之和为该子问题的代价，所以整个递归树从根节点到叶子节点代价之和为递归方程总代价

迭代（递归树）方法

例1. $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) = 3T(\lfloor n/4 \rfloor) + cn^2$



计算代价分为两部分：
1）计算内部节点代价，即分解和合并子问题的代价总和，这部分一般呈现等比数列求和
2）计算叶子节点代价总和，即分解到最后的边界条件值，关键在于求解叶子节点个数

最底层叶子节点数：$3^{\log_4^n} = n^{\log_4^3}$，叶子节点总代价：$n^{\log_4 3}T(1) = \Theta(n^{\log_4 3})$

总的代价之和=迭代树右侧所有代价之和！！！

迭代（递归树）方法

例1. $T(n) = 3T(\lfloor n/4 \rfloor + \Theta(n^2))$

$$T(n) = cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$= \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$= \frac{(3/16)^{\log_4 n} - 1}{(3/16)-1} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

根据：$\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$

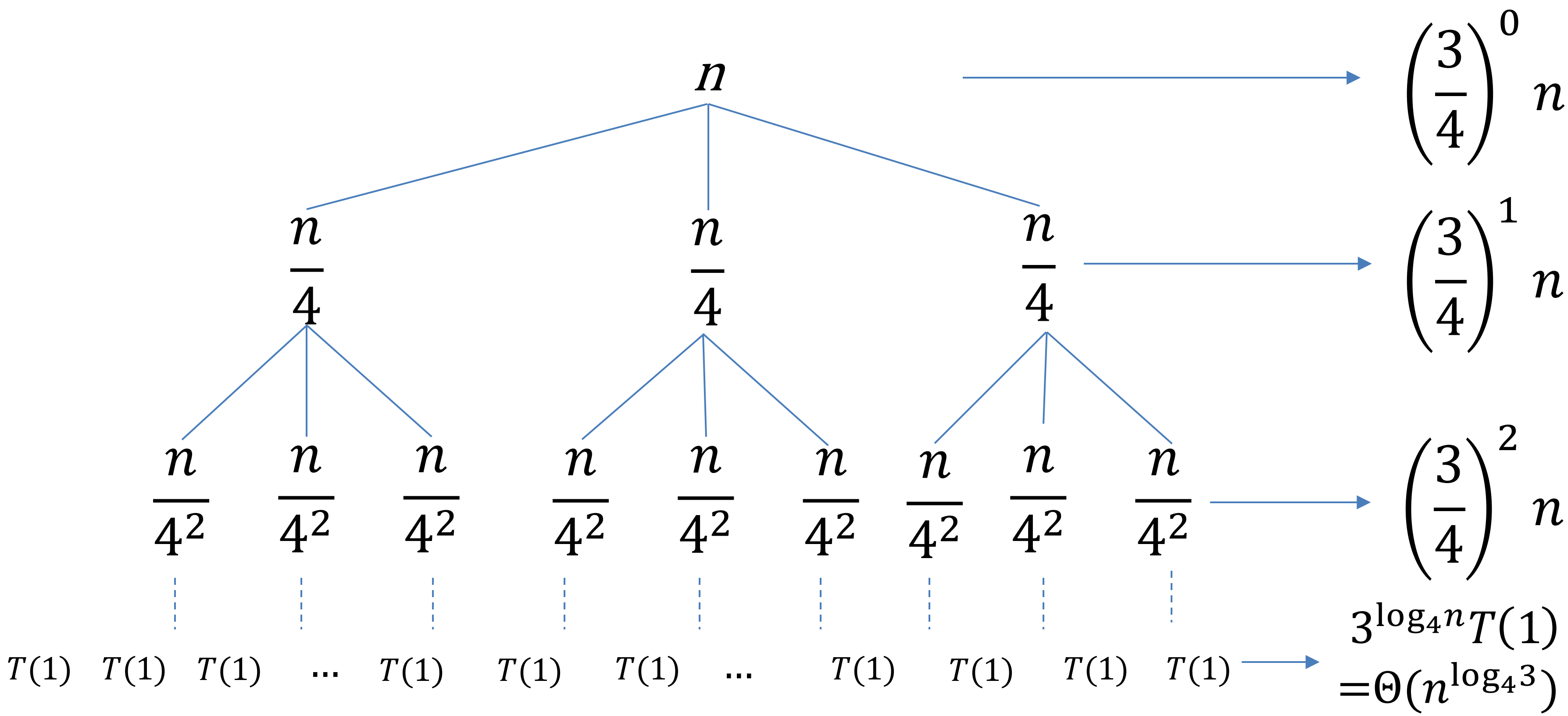
$$< \frac{16}{13} cn^2 + \Theta\left(n^{\log_4 3}\right),\ 当 n \to \infty$$

我们猜测T(n)=O(n²),然后用归纳法证明

迭代递归树方法提供了一种简单直接的猜想界的方法。

迭代（递归树）方法

例2. $T(n) = n + 3T(\lfloor {}^{n}/_{4} \rfloor)$



迭代（递归树）方法

例 1. $T(n)=n+3T\left(\left\lfloor n/4 \right\rfloor\right)$

$$=n+3\left(\left\lfloor n/4 \right\rfloor+3T\left(\left\lfloor n/16 \right\rfloor\right)\right)$$

$$=n+3\left(\left\lfloor n/4 \right\rfloor+3\left(\left\lfloor \frac{n}{16} \right\rfloor+3T\left(\left\lfloor n/64 \right\rfloor\right)\right)\right)$$

$$=n+3\left\lfloor n/4 \right\rfloor+9\left\lfloor \frac{n}{16} \right\rfloor+27T\left(\left\lfloor n/64 \right\rfloor\right)$$

$$=n+3\left\lfloor n/4 \right\rfloor+3^2\left\lfloor \frac{n}{4^2} \right\rfloor+3^3\left(\left\lfloor \frac{n}{4^3} \right\rfloor\right)+\ldots\ldots+3^i\,T\left(\left\lfloor n/4^i \right\rfloor\right)$$

令 $\frac{n}{4^i}=1 \Rightarrow 4^i=n \Rightarrow i=\log_4 n$

$$=n+3\left\lfloor n/4 \right\rfloor+3^2\left\lfloor \frac{n}{4^2} \right\rfloor+3^3\left(\left\lfloor \frac{n}{4^3} \right\rfloor\right)+\ldots\ldots+3^{\log_4 n}\,T\left(\left\lfloor 1 \right\rfloor\right)$$

$$\le \sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i n+O(n)\le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i+O(n)=n\times\frac{1}{1-3/4}+O(n)=4n+O(n)=O(n)$$

Master定理（主方法）

目的：求解递归式$T(n) = aT(n/b) + f(n)$的复杂度，
$a \geq 1, b > 1$是常数。

记住三种情况，则不需笔纸，可以用Master定理直接求解。

Master定理

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

Master定理适用于，通过解一个问题的子问题，来解一个问题，并且**子问题规模相同**

a ：子问题数量

b ：子问题从原问题缩小的比例，n/b为子问题规模

f(n)：将问题分解和子问题解整合的代

包括两项：$aT(n/b)$ 和 $f(n)$，关键看哪项带来的增长更快？

$aT(n/b)$带来的增长阶数可以等价用$n^{\log_b a}$ 表示，结合上一节中递归树求解实例，$n^{\log_b a}$ 等价于递归树中最底层的叶子节点个数（具体看算法导论中4.6节的基于递归树的证明），因此所有叶节点的代价总和为$n^{\log_b a}\ T(1) = \Theta(n^{\log_b a})$，而内部节点的代价求和与f(n)相关。

Master定理

Master 定理：设 $a \geq 1$和$b > 1$是常数， $f(n)$是一个函数，$T(n)$是定义在非负整数集上的函数

$$T(n) = aT(n/b) + f(n)。$$

那么$T(n)$可以如下求解：

（1）若$f(n) = O(n^{\log_b a-\varepsilon})$，$\varepsilon > 0$是常数，则$T(n) = \Theta(n^{\log_b a})$。

（2）若$f(n) = \Theta(n^{\log_b a})$（**同阶**），则$T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(f(n)\lg n)$。

（3）若$f(n) = \Omega(n^{\log_b a+\varepsilon})$，$\varepsilon > 0$是常数，且对某个常数$c < 1$和所有充分大的$n$，有$af(n/b) \leq cf(n)$，则$T(n) = \Theta(f(n))$。

正则条件

怎么证明？《算法导论》中4.6节采用递归树证明。

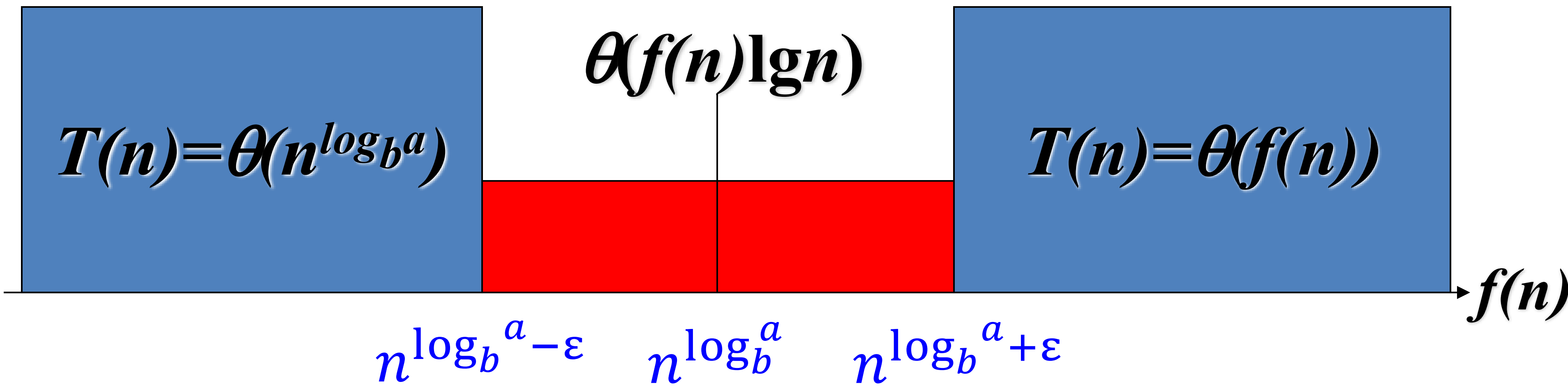
和直觉一致

直观地：我们用$f(n)$与$n^{\log_b a}$比较：

（1）若$n^{\log_b a}$多项式地大$(f(n)=O(n^{\log_b a-\varepsilon}))$，则$T(n)=\Theta(n^{\log_b a})$。

（2）若$f(n)$多项式地大$(f(n)=\Omega(n^{\log_b a+\varepsilon}))$，则$T(n)=\Theta(f(n))$ 不要忘了正则条件

（3）若$f(n)$与$n^{\log_b a}$同阶，则$T(n)=\Theta(n^{\log_b a}\lg n)=\Theta(f(n)\lg n)$。



对于红色部分或者情况3不满足正则条件，Master定理无能为力

Master定理

更进一步：

（1）在第一种情况，$f(n)$不仅小于$n^{\log_b^a}$，必须多项式地小于，即对于一个常数$\varepsilon > 0$，$f(n) = O\left(n^{\log_b{}^a - \varepsilon}\right)$。

（2）在第三种情况，$f(n)$不仅大于$n^{\log_b^a}$，必须多项式地大于，即对于一个常数$\varepsilon > 0$，$f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$。

Master定理

（1）若$f(n)=O(n^{\log_b^a-\varepsilon})$，$\varepsilon>0$是常数，则$T(n)=\Theta\left(n^{\log_b^a}\right)$

（2）若$f(n)=\Theta(n^{\log_b^a})$，则$T(n)=\Theta(n^{\log_b^a}\lg n)$。

例1. 求解$T(n)=9T(n/3)+n$。

a, b值不要搞错

解：$a=9$，$b=3$，$f(n)=n$，$n^{\log_b^a}=\Theta(n^2)$

因为$f(n)=n=O(n^{\log_b^a-\varepsilon})$，$\varepsilon=1$

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

例2. 求解$T(n)=T(2n/3)+1$。

解：$a=1$，$b=3/2$，$f(n)=1$，$n^{\log_b{}^a}=n^{\log_{3/2}{}^1}=n^0=1$，

因为$f(n)=1=\Theta(1)=\Theta(n^{\log_b{}^a})$，

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

Master定理

（3）若$f(n) = \Omega(n^{\log_b^a+\varepsilon})$，$\varepsilon > 0$是常数，且对于所有充分大的$n$，$af(^n/_b) \le cf(n)$，$c < 1$是常数，则$T(n) = \Theta(f(n))$。

例3. 求解 $T(n) = 3T(^n/_4) + n\lg n$ 。

解：$a = 3$，$b = 4$，$f(n) = n\lg n$，$n^{\log_b^a} = n^{\log_4^3} = O(n^{0.793})$

（1）$f(n) = n\lg n \ge n^{\log_b^a+\varepsilon}$，$\varepsilon \approx 0.2$

（2）对所有n，$af(^n/_b) = 3 \times \frac{n}{4}\lg\frac{n}{4} = \frac{3}{4}n\lg\frac{n}{4} \le \frac{3}{4}n\lg n = cf(n)$，$c = \frac{3}{4}$

于是$T(n) = \Theta(f(n)) = \Theta(n\lg n)$。

不要忘了正则条件

Master定理

例4. 求解$T(n) = 2T(^n/_2) + n\lg n$。

解： $a = 2$，$b = 2$，$f(n) = n\lg n$，$n^{\log_b^a} = n^{\log_2^2} = n$

$f(n) = n\lg n$大于$n^{\log_b^a} = n$，但不是多项式地大于，即不能找到$\varepsilon > 0$，使得$n\lg n = \Omega(n^{\log_b^a+\varepsilon})$，因此Master 定理不适用于该$T(n)$。


Thank You