第四讲
算法引论

主讲教师：朱建明
Email: jmzhu@ucas.ac.cn
Homepage: http://people.gucas.ac.cn/~jianming

- ✓ 算法与程序
- ✓ 算法复杂性分析

example

算法与程序

算法：是满足下述性质的指令序列。

- 输　入：有零个或多个外部量作为算法的输入。
- 输　出：算法产生至少一个量作为输出。
- 确定性：组成算法的每条指令清晰、无歧义。
- 有限性：算法中每条指令的执行次数有限，执行每条指令的时间也有限。

程序：是算法用某种程序设计语言的具体实现。

程序可以不满足算法的性质(4)即有限性。

现代优化计算方法

算法的描述

- 一个算法可以用自然语言、计算机程序语言或其它语言来说明，惟一的要求是该说明必须精确地描述计算过程。
- 一般而言，描述算法最合适的语言是介于自然语言和程序语言之间的伪语言。
- 从易于上机验证算法和提高实际程序设计能力考虑，采用C语言描述算法。

博学笃志 格物明德

算法的正确性

- 若一个算法对于每个输入实例均能终止并给出正确的结果，则称该算法是正确的。
- 正确的算法解决了给定的计算问题。
- 一个不正确的算法是指对某些输入实例不终止，或者虽然终止但给出的结果不是所渴望得到的答案，一般只考虑正确的算法

算法分析

- **评价算法好坏的标准**

　　求解同一计算问题可能有许多不同的算法，究竟如何来评价这些算法的好坏以便从中选出较好的算法呢？

　选用的算法首先应该是"正确"的。此外，主要考虑如下三点：

① 执行算法所耗费的时间；

② 执行算法所耗费的存储空间，其中主要考虑辅助存储空间；

③ 算法应易于理解，易于编码，易于调试等等

算法性能选择

一个占存储空间小、运行时间短、其它性能也好的算法是很难做到的。原因是上述要求有时相互抵触：要节约算法的执行时间往往要以牺牲更多的空间为代价；而为了节省空间可能要耗费更多的计算时间。因此我们只能根据具体情况有所侧重：

① 若该程序使用次数较少，则力求算法简明易懂；

② 对于反复多次使用的程序，应尽可能选用快速的算法；

③ 若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省空间。

著名公式

Algorithm + Data Structure = Programming

好的算法

提高求解问题的效率

节省存储空间

需要解决的问题

问题→一个求解算法： 算法设计技术

算法→算法的评价： 算法分析技术

算法复杂性分析

- 算法复杂性是算法运行所需要的计算机资源的量
  - 需要时间资源的量称为**时间复杂性**，
  - 需要的空间资源的量称为**空间复杂性**。
- 这个量应该只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。

- 如果分别用N、I和A表示算法要解问题的规模、算法的输入和算法本身，而且用C表示复杂性，那么，应该有C=F(N,I,A)。
- 一般把时间复杂性和空间复杂性分开，并分别用T和S来表示，则有： T=T(N,I)和S=S(N,I) 。

（通常，让A隐含在复杂性函数名当中）

现代优化计算方法

算法时间复杂性

最坏情况下的时间复杂性：

$$T_{\max}(N) = \max_{I \in D_N} T(N,I) = \max_{I \in D_N} \sum_{i=1}^{k} t_i e_i(N,I) = \sum_{i=1}^{k} t_i e_i(N,I^*) = T(N,I^*)$$

最好情况下的时间复杂性：

$$T_{\min}(N) = \min_{I \in D_N} T(N,I) = \min_{I \in D_N} \sum_{i=1}^{k} t_i e_i(N,I) = \sum_{i=1}^{k} t_i e_i(N,\tilde{I}) = T(N,\tilde{I})$$

平均情况下的时间复杂性：

$$T_{\text{avg}}(N) = \sum_{I \in D_N} P(I) T(N,I) = \sum_{I \in D_N} P(I) \sum_{i=1}^{k} t_i e_i(N,I)$$

其中D_N是规模为N的合法输入的集合；I*是D_N中使T(N, I*)达到$T_{\max}$(N)的合法输入；$\tilde{I}$是中使T(N, $\tilde{I}$)达到$T_{\min}$(N)的合法输入；而P(I)是在算法的应用中出现输入I的概率。

格物博学 明德笃志

例：　搜索问题

输入：非降顺序排列的数组L，元素数为n．数x

输出：j．若x在L中，j是x首次出现的序标；

否则j = 0

算法　顺序搜索

假设　x在L中的概率为p

x在L中不同位置是等概分布的，则

$$W(n) = n$$

$$A(n) = \sum_{i=1}^{n} i\frac{p}{n} + (1-p)n = \frac{p(n+1)}{2} + (1-p)n$$

渐近复杂性

- ✓ 渐近复杂性：对于T(N)，如果存在$\tilde{T}(N)$,使得

$$\lim_{N\to\infty}\frac{T(N)-\tilde{T}(N)}{T(N)}=0$$

,则 $\tilde{T}(N)$ 为T(N)的渐近性态。

- ✓ 渐近记号: O,，Ω，Θ，o

算法复杂性在渐近意义下的阶

渐近意义下的记号：O、Ω、θ、o
设f(N)和g(N)是定义在正数集上的正函数。

O的定义：如果存在正的常数C和自然数N_0，使得当$N \geq N_0$时有$f(N) \leq Cg(N)$，则称函数f(N)当N充分大时上有界，且g(N)是它的一个**上界**，记为$f(N)=O(g(N))$。即f(N)的阶不高于g(N)的阶。

运算规则

根据O的定义，容易证明它有如下运算规则：

(1)$O(f)+O(g)=O(\max(f,g))$；

(2)$O(f)+O(g)=O(f+g)$；

(3)$O(f)O(g)=O(fg)$；

(4)如果$g(N)=O(f(N))$，则$O(f)+O(g)=O(f)$；

(5)$O(Cf(N))=O(f(N))$，其中C是一个正的常数；

(6)$f=O(f)$。

函数的阶，下界

Ω的定义：如果存在正的常数C和自然数N_0，使得当$N \geq N_0$时有$f(N) \geq Cg(N)$，则称函数$f(N)$当N充分大时下有界，且$g(N)$是它的一个**下界**，记为$f(N)=\Omega(g(N))$。即$f(N)$的阶不低于$g(N)$的阶。

现代优化计算方法

同阶

θ 的定义： 定义 $f(N)=\theta(g(N))$ 当且仅当 $f(N)=O(g(N))$ 且 $f(N)=\Omega(g(N))$。此时称 $f(N)$ 与 $g(N)$ 同阶

博学笃志 格物明德

低阶

o的定义：对于任意给定的 $\varepsilon>0$，都存在正整数N0，使得当N≥N0时有 $f(N)/Cg(N)\le\varepsilon$，则称函数f(N)当N充分大时的阶比g(N)低，记为 $f(N)=o(g(N))$。

例如，$4N\log N+7=o(3N^2+4N\log N+7)$。

证明例子 $f(n)=\frac{1}{2}n^2-3n,$

证明 $f(n)=0(n^2)$

记住定义：$N\geq N_0$ 时有 $f(N)\leq Cg(N)$ 则 **f(n)=O（g(n)）**

此处：**g(n)** ＝n^2

证 ：取 c=1，n_0=1，当$N>n_0$时满足 $f(N)\leq Cg(N)$

则 **f(n)=O（g(n)）**

即 **$f(n)=O(n^2)$**

函数阶的定义（极限定义）

根据 x＝ $\lim\limits_{N\to\infty}\dfrac{f(N)}{g(N)}$ 所得的结果

- ✓ 1） x ＜∞, 包括＝0 则f ∈O(g)
- ✓ 2） x > 0,包括 ＝∞ 则f ∈ Ω(g)
- ✓ 3）0 ＜ x ＜∞ 则f ∈ θ (g)
- ✓ 特殊情况1 x= 0 then f ∈ o(g)， f比g低阶
- ✓ 特殊情况2 x= ∞ then f ∈ ω(g)， f比g高阶