吉林大学

第二章 导引与基本数据结构

目录

2.1 算法

- 什么是算法
- 算法的五个重要特性
- 计算过程与算法的区别
- 问题的求解过程
- 算法学习的基本内容

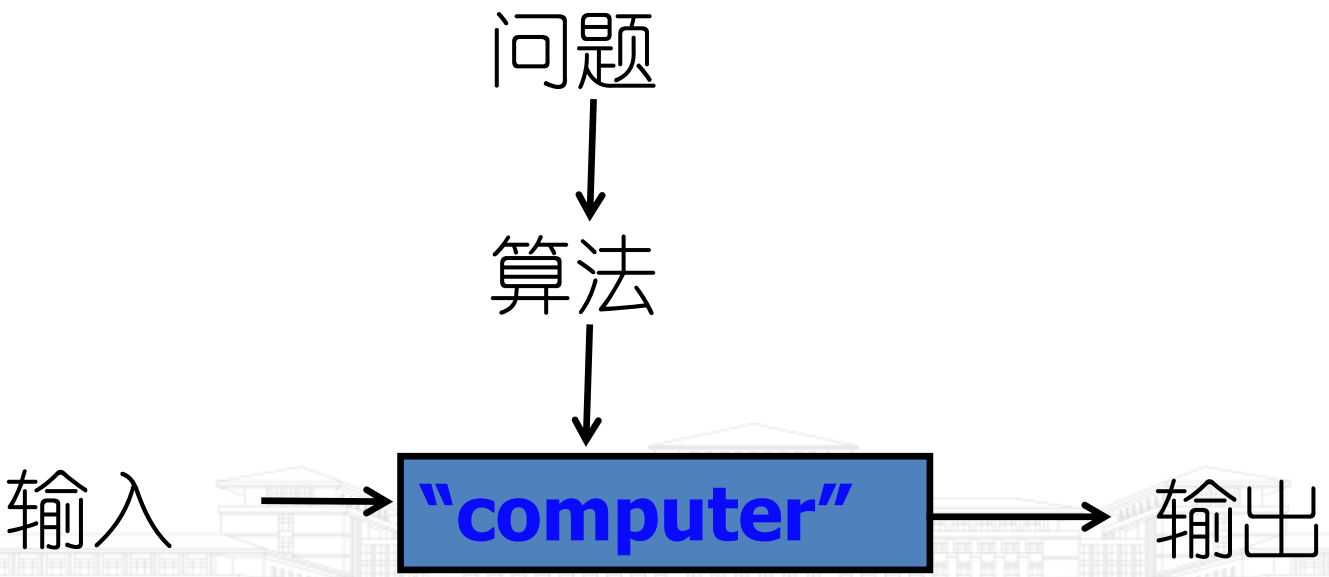
什么是算法

- 对于这个概念，没有一个大家公认的定义。
- 算法的非形式描述：算法就是一组有穷的规则，它规定了解决某一特定类型问题的一系列运算。

Knuth给出的概念

什么是算法？

- 对算法的基本共识：算法是一系列解决问题的明确指令，也就是说，对于符合一定规范的输入，能够在有限时间内获得要求的输出。



算法(Algorithm)的五个重要特性

- 满足如下五条特性的一组规则才能被称为算法
  - 有穷性(Finiteness)
  - 确定性 (Definiteness)
  - 输入 (Input)
  - 输出(Output)
  - 可行性 (Effectiveness)

欧几里得算法

The Euclidean Algorithm

Procedure Euclid (m, n)

// 求正整数m和n (m≥n)的最大公因数

r ← Mod(m,n)

While r≠0 Do

m ←n. n ←r. r ←mod(m,n).

Repeat

Return n

End Euclid

1. 有穷性 （Finiteness）

- 一个算法必须在有限步骤之后终止。
- 在Euclid算法中，对于任何给定的n，随着算法的推进n不断减小，n的变化是一个非0整数的递减序列， Euclid算法必然会在有限步骤后终止。
- 如果m和n的值非常大呢？
  - 也仅仅是增加了算法的步骤数，Euclid算法仍然会在有限步骤之后终止

2. 确定性 （Definiteness）

- 一个算法的每个步骤都必须精确定义；要执行的每个动作都必须严格地、无歧义地描述清楚
- 在Euclid算法中，因为m和n约定为正整数，所以求余操作mod(m,n)是确定的。如果没有正整数的约定呢？
- 求余操作就是不确定的。Mod(-10, 2.189)是什么？Mod(598,0)是什么？不确定！！！

3. 输入（input）

- 一个算法有0个或多个输入，是在算法开始之前最初赋给它的量
- 这些输入是从特定的对象集合中取出的
- Euclid算法中，有几个输入？
- 两个输入：m和n，且二者都是从正整数集合中取出的

4. 输出（output）

- 一个算法有一个或多个输出：这些输出是和输入有特定关系的量
- Euclid算法的输出是什么？
- 有一个输出，是两个输入（m和n）的最大公因数

对算法的两种理解

- 算法就是能够对于一些给定的输入数据进行一系列的操作，将它们转化成输出的处理过程。
- 算法就是解决一个定义好的问题的工具，问题的定义详细说明了输入和输出的关系。

5. 可行性（Effectiveness）

- 一个算法被认为是可行的，是指它的所有运算都是基本的运算，理论上可用纸和笔在有限时间内精确完成
- Euclid算法中涉及的运算包括
  - 整数相除求余
  - 检查一个整数是否为0
  - 赋值运算

全是基本运算，可在有限时间内精确完成

如果运算涉及的值是有无穷小数的实数，同样的运算就不具有可行性

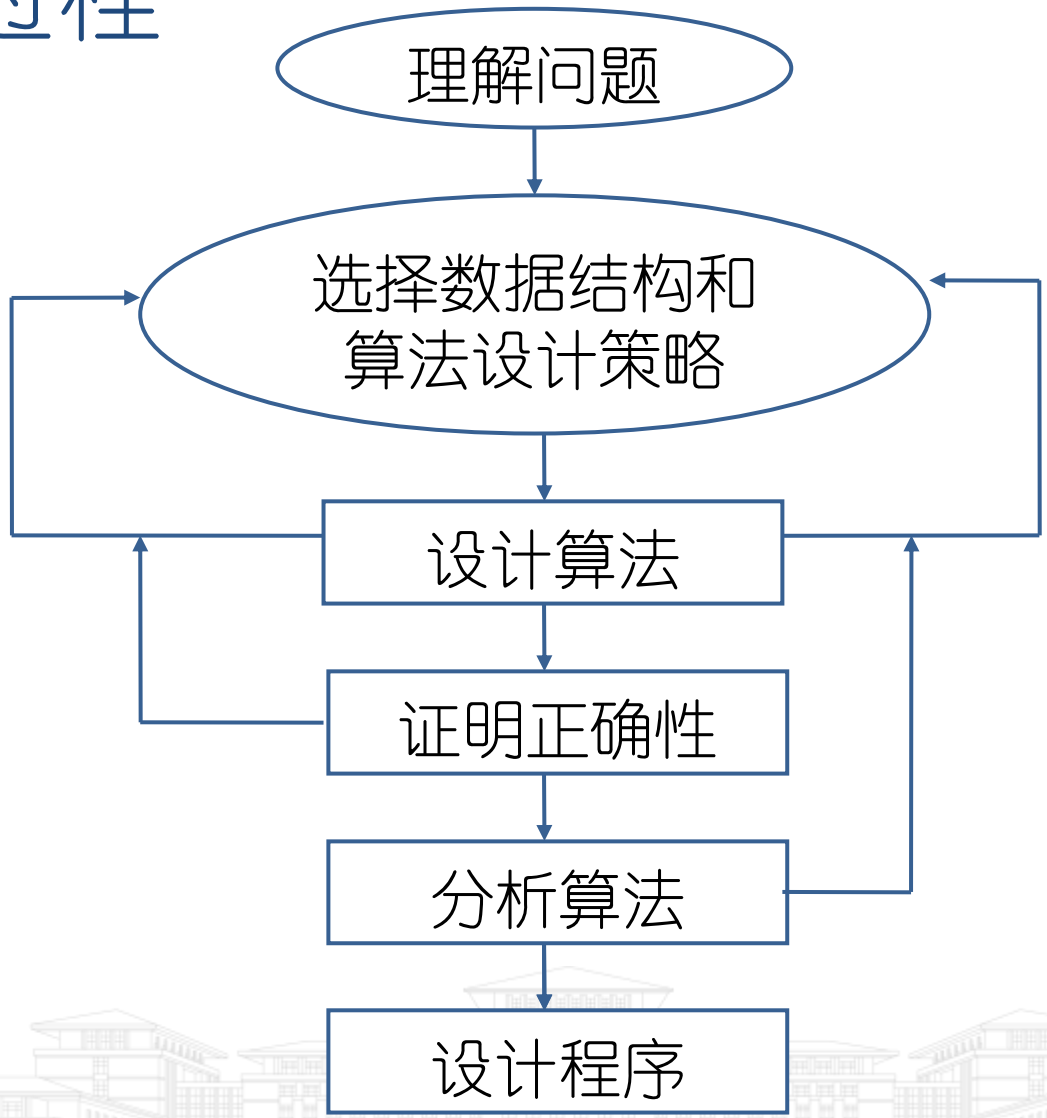
对有穷性的进一步说明

- 要说明的是，就实用而言，有穷性的限制实际上不够强。
- 一个有用的算法，不仅要求步骤有限，而且要求非常有限的、合理的步骤数。

计算过程与算法的区别

- 计算过程可以不满足算法的性质有穷性。
- 例如操作系统，当没有任务时，操作系统并不终止，而是处于等待状态，直到有新的任务启动，因而不是一个算法。
- 程序 = 算法 + 数据结构 (By Niklaus Wirth)

问题的求解过程



算法学习的基本内容

- 如何设计算法
- 如何表示算法
- 如何确认算法
- 如何分析算法
- 如何测试程序

如何设计算法

- 面对具体问题，运用一些基本设计策略，规划算法。
  - 被实践证明是有用的
  - 计算机科学、运筹学、电器工程等领域
  - 设计出很多精致有效的好算法
- 掌握这些策略，设计出更多的好算法

如何表示算法

- 设计的算法要用恰当的方式地表示出来
- 采用结构程序设计的方式
- SPARKS程序设计语言——简单明了

如何确认算法

- 算法确认(algorithm validation)——证明该算法对所有可能的合法输入，都能给出正确答案
- 算法确认的目的——确保该算法能正确无误地工作
  - 穷举法
  - 推理——数学归纳法、反证法
  - 构造性证明
  - 测试

如何分析算法

- 分析执行一个算法时，
  - 占用多少CPU时间完成运算；
  - 占用多少存储器的存储空间，存放程序和数据。
- 即对一个算法需要多少计算时间和存储空间，做定量分析。

测试程序

- 调试——调试只能指出有错误，而不能指出它们不存在错误。
  - 源于程序正确性的证明还没有取得突破性进展。
- 时空分布图
  - 用各种给定数据执行调试后的程序
  - 测定计算时间和空间
  - 印证之前的分析是否正确
  - 指出实现最优化的有效逻辑位置

2.2 分析算法

- 算法分析目的
- 算法分析的准备工作
- 计算时间的渐进表示
- 一些证明方法
- 作时空性能分布图

算法分析的目的

- 算法分析(analysis of algorithms)是对一个算法需要多少计算时间和存储空间作定量的分析。

  ——确定算法在什么样的环境下能够有效地运行。

- 分析在最好、最坏和平均情况下的执行情况。

  ——对同一问题不同算法的有效性作出比较。

时间复杂性的形式化定义

- 算法的时间复杂性$T(n)$;
- 算法的空间复杂性$S(n)$;
- 其中n是问题的规模。

最坏情况下： $T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \}$

最好情况下： $T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \}$

平均情况下： $T_{\text{avg}}(n) = \sum_{size(I)=n} p(I)T(I)$

其中I是问题的规模为n的实例，

$p(I)$是实例I出现的概率。

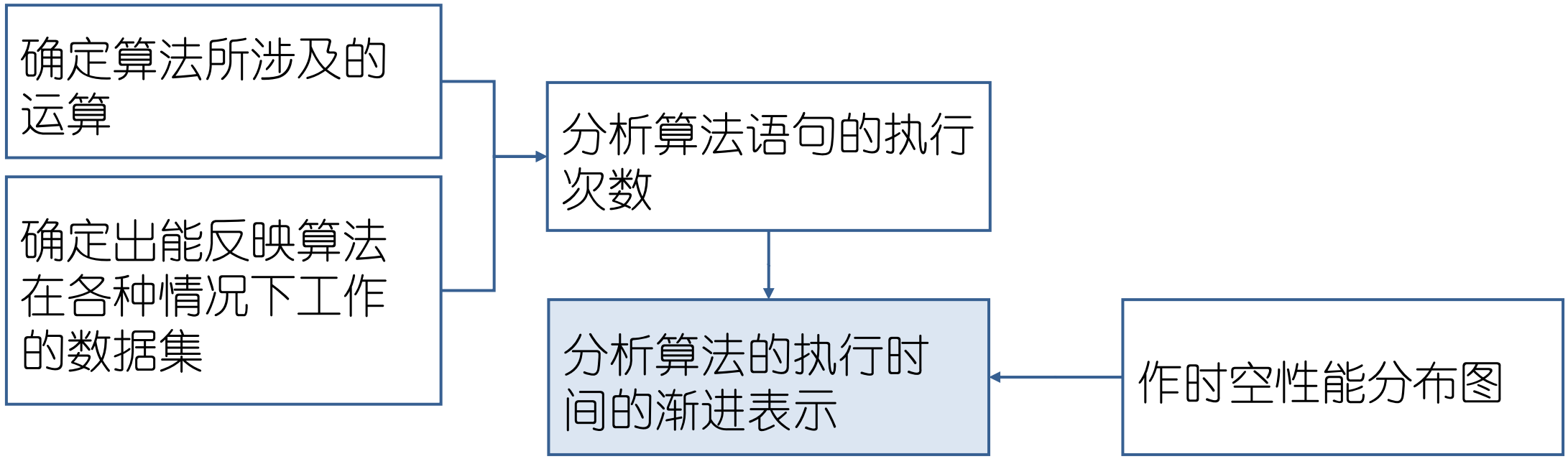
算法运行假定的计算机类型

- 要求独立于具体的软硬件环境单纯分析算法。
- 假定使用一台通用计算机
  - 顺序处理每条指令；
  - 存储容量足够大；
  - 存取时间恒定。

算法分析过程

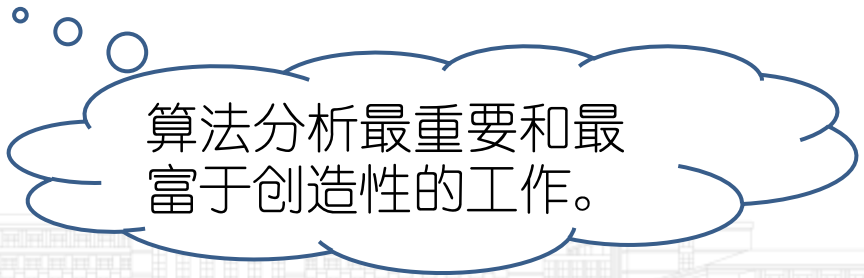
全面分析的两个阶段

准备工作　　事前分析　　事后测试



准备工作

- 首先确定使用哪些运算以及执行这些运算所用的时间。
  - 基本运算
  - 由一些更基本的任意长序列的运算所组成的复杂运算
- 其次是要确定出能反映算法在各种情况下工作的数据集。
  - 编造出能产生最好、最坏和有代表性情况的数据配置
  - 通过使用这些数据来运行算法，以更好地了解算法的性能



基本运算

- 时间囿界于常数的运算
  - 加、减、乘、除整数算术运算
  - 浮点算术、字符比较、对变量赋值、过程调用等
- 每种运算所用时间虽然不同，但都只花费一个固定量的时间

复杂运算

- 由一些更基本的任意长序列的运算组成，如：两个字符串的比较运算
  - 一系列字符比较指令
  - 一个字符的比较时间——囿界于常数
  - 字符串比较的时间总量则取决于字符串的长度

全面分析算法的两个阶段

- 事前分析(a priori analysis)
  - 确定每条语句的执行次数
  - 求出该算法的一个时间限界函数(关于问题规模n的函数)
- 事后测试(a posterior testing)
  - 作时空性能分布图

运行时间的度量单位

- 方法1：使用时间的标准度量单位（秒、毫秒）来度量算法程序的运行时间。

其明显缺陷

- 依赖于特定计算机的运行速度；
- 依赖于算法程序实现的质量；
- 依赖于使用哪种编译器将程序转化为机器语言；
- 对实际运行时间进行计时也比较困难。

例：计算语句x←z+y在下面三个程序段中的执行次数

程序段1

```
begin
x←z+y
end
```

1次

程序段2

```
begin
for i←1 to n do
      x←z+y
end
```

n次

程序段3

```
begin
for  i←1 to n do
   for  j←1 to n do
         x←z+y
end
```

n^2次

运行时间的度量单位

- 方法2：统计算法每一步操作的执行次数。

  缺点：过于困难、且没有必要

只需要:

(1)找出算法中最重要的操作（对总运行时间贡献最大的操作）——基本操作（basic operation）；

(2)然后计算在该算法中基本操作运行的次数—引入渐进表示

欧几里得算法

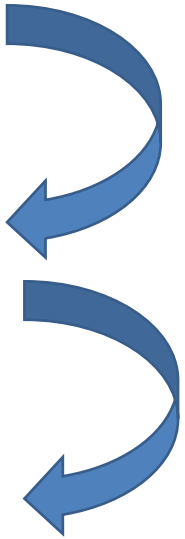
The Euclidean Algorithm

```
Procedure Euclid (m, n)
        // 求正整数m和n (m≥n)的最大公因数
        r ← Mod(m,n)
         While r≠0 Do
                m ←n. n ←r. r ←mod(m,n).
        Repeat
        Return n
End Euclid
```

不难发现算法中的基本操作：通常是算法最内层循环中最费时的操作

总结

算法的计算时间



~~语句的执行时间~~

每一步操作的执行次数

~~低次项(数量级低的语句)~~

算法的渐进表示

准确分析出每一步操作的执行次数是很困难的，因此我们对事前分析的计算时间进行渐进表示。

计算时间的渐进表示

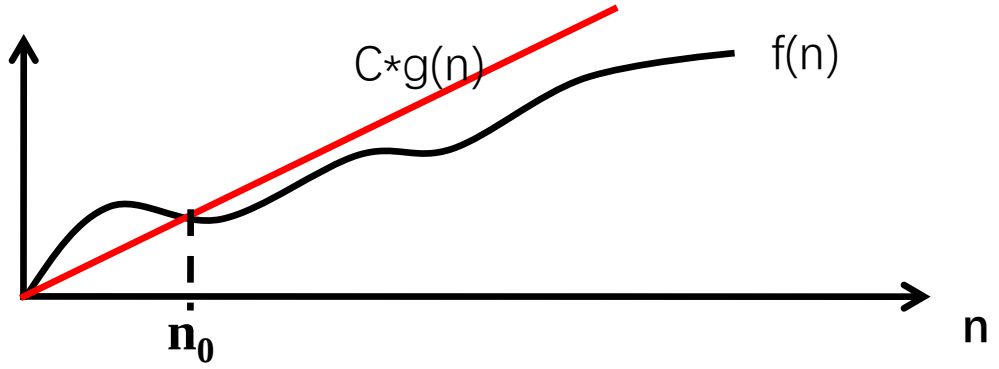
- 定义2.1：f(n)= O(g(n))
- 定义2.2：f(n)=Ω(g(n))
- 定义2.3：f(n)=Θ(g(n))
- 定理2.1

- n表示问题规模，
  - 输入或输出量；
  - 两者之和；
  - 其中之一的某种测度。
- f(n) 表示算法的计算时间。
- g(n)是在事前分析中确定的某个形式简单的函数。

f(n)与机器和语言有关，而g(n)是独立于机器和语言的。

定义2.1

$f(n) \in O(g(n))$



如果存在两个正常数c和n_0，对于所有的$n \geq n_0$，有$|f(n)| \leq c|g(n)|$，则记作：$f(n)=O(g(n))$，称g(n)是f(n) 的一个渐进上限。

- 当n充分大时，f(n)有上界，一个常数倍的g(n)是f(n)的一个上界，f(n)的数量级就是g(n)。
- f(n)的阶不高于g(n)的阶。
- 在确定f(n)的数量级时，总是试图求出最小的g(n)。
- 有关证明中，找出c和n_0是关键。

判断$f(n) = O(g(n))$？

基于定义证明你的判断

- $f(n) = 3n, \quad g(n)=n$
- $f(n) = n+1024, \quad g(n)=1025n$
- $f(n) = 2n^2+11n-10, \quad g(n)=3n^2$
- $f(n) = n^2, \quad g(n)= n^3$
- $f(n) = n^3, \quad g(n)= n^2$

判断f(n) ＝O(g(n)) ？

存在正常数c和n_0，对于所有的$n \geq n_0$，有$|f(n)| \leq c|g(n)|$

- 例1. f(n) ＝3n,　g(n)=n

  是否存在c和n_0，对于所有的$n \geq n_0$，满足$3n \leq cn$,

  只需c=3, n_0 =1

  证明：∃ c=3, n_0 =1, ∀ $n \geq n_0$，均有$3n \leq cn$，　故f(n) ＝O(g(n)) .

- 例5. **f(n) ＝ n^3，　g(n)= n^2**

  分析：对于任何正常数c和n_0，存在$n_1 \geq n_0$，使得$|f(n_1)| > c|g(n_1)|$

  往证：∀正常数c和n_0，　∃ $n_1 \geq n_0$，使得$n_1^3 > cn_1^2$

  只需n_1>c即可

  证明：∀正常数c和n_0，　∃ n_1= max{n_0 , c+1}

  使得$n_1^3 > cn_1^2$，故$n^3 \neq O(n^2)$. 证毕。

O性质

对于非负的f(n)和g(n)，根据定义2.1，有如下性质：

1.$O(f(n))+O(g(n)) = O(\max(f(n), g(n))$；

2.$O(f(n))+O(g(n)) = O(f(n)+g(n))$；

3.$O(f(n))\ O(g(n)) = O(f(n)\ g(n))$；

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

5.$O(cf(n)) = O(f(n))$，其中c是一个正的常数；

6.$f(n) = O(f(n))$。

证明$O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$

不妨设p(n)= max{f(n),g(n)}

设f’(n)=O(f(n))，则存在c_1,n_1，当$n\geq n_1$时，$f'(n) \leq c_1*f(n)$

设g’(n)=O(g(n))，则存在c_2,n_2，当$n\geq n_2$时，$g'(n) \leq c_2*g(n)$

则O(f(n))+O(g(n)) =f’(n)+g’(n)

当$n \geq \max\{n_1,n_2\}$时，

$f'(n)+g'(n) \leq c_1*f(n)+ c_2*g(n) \leq c_1*p(n)+ c_2*p(n)=(c_1+c_2)*p(n)$

即存在$c_3=c_1+c_2$，$n_3= \max\{n_1,n_2\}$，当$n \geq n_3$时，$f'(n)+g'(n) \leq c_3*p(n)$

故$O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$

定理2.1

若$A(n)=a_m n^m+\ldots+a_1 n+a_0$是一个m次多项式，则$A(n)=O(n^m)$。

证明：取$n_0=1$，当$n\geq n_0$时

由A(n)的定义和不等式关系$|A+B| \leq |A|+|B|$有

$$|A(n)| = |a_m n^m+\ldots+a_1 n+a_0| \leq |a_m| n^m+\ldots+|a_1| n+|a_0|$$

$$= (|a_m|+|a_{m-1}|/n \ \ldots+|a_0|/n^m)\, n^m$$

$$\leq (|a_m|+|a_{m-1}|\ldots+|a_0|)\, n^m$$

取 $c= |a_m|+|a_{m-1}|\ldots+|a_0|$，有$|A(n)| \leq cn^m$

即：$A(n)=O(n^m)$，定理得证。

定理2.1：若$A(n)=a_m n^m+\ldots+a_1 n+a_0$是一个m次多项式，则$A(n)=O(n^m)$。

- 定理2.1表明，变量n的最高阶数为m的任一多项式，与n^m同阶。因此一个计算时间为m阶多项式的算法，其时间都可以用$O(n^m)$来表示。
- 若一个算法有数量级为$c_1 n^{m1}$，$c_2 n^{m2}$，… $c_k n^{mk}$的k个语句，则算法的数量级及计算时间就是

$$c_1 n^{m1}+c_2 n^{m2}+\ldots+c_k n^{mk}=O(n^m),\ m=\max\{m_i|1\le i\le k\}$$

数量级对算法有效性的影响

- 从计算时间上算法可以分为两类：
  - 多项式时间算法(polynomial time algorithm):用多项式来对其计算时间限界的算法

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$$

  - 指数时间算法(exponential time algorithm):计算时间用指数函数限界的算法

$$O(2^n)<O(n!)<O(n^n)$$

不同时间复杂性函数的对比

T(n) 微秒	logn	n	nlogn	n^2	n^3	n^5	2^n	3^n	n!
n=10	3.3	10	33	100	1 毫秒	0.1 秒	1 毫秒	59 毫秒	3.6 秒
n=40	5.3	40	213	1600	64 毫秒	1.3 分	12.7 天	3855 世纪	10^3 世纪
n=60	5.9	60	354	3600	216 毫秒	7.6 分	366 世纪	1.3×10^{13} 世纪	10^{66} 世纪

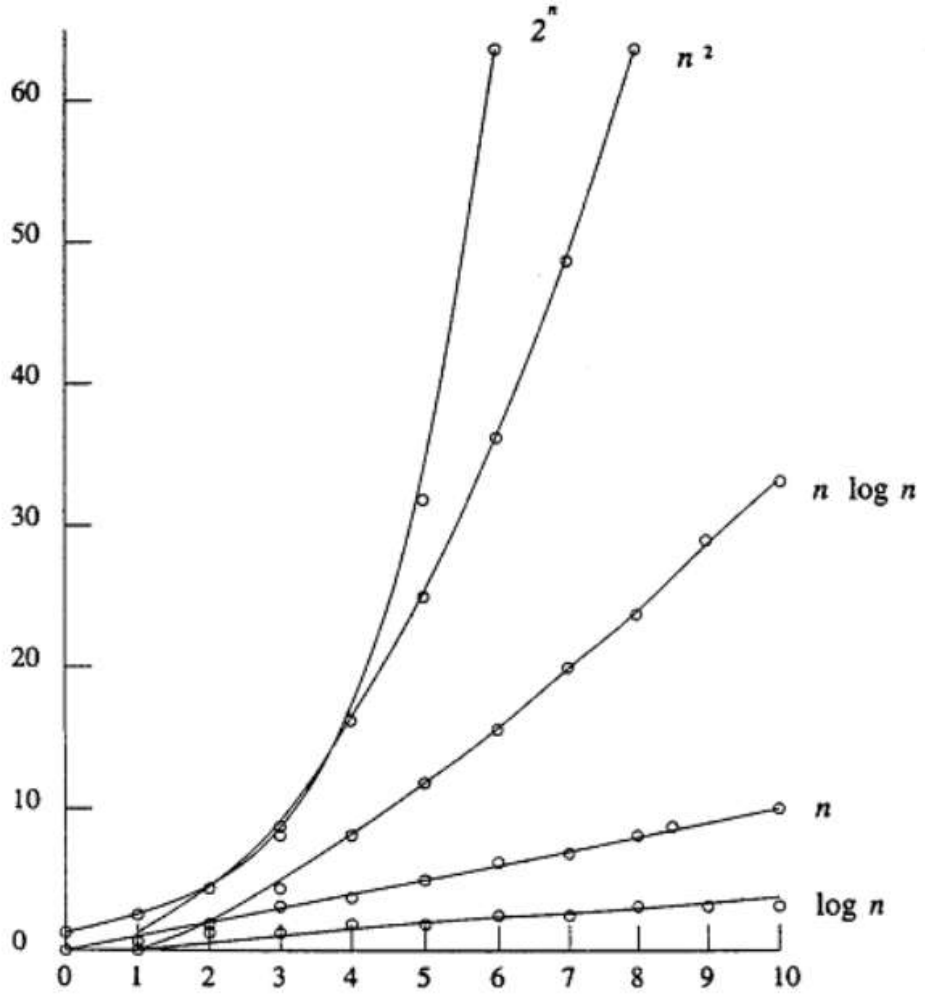
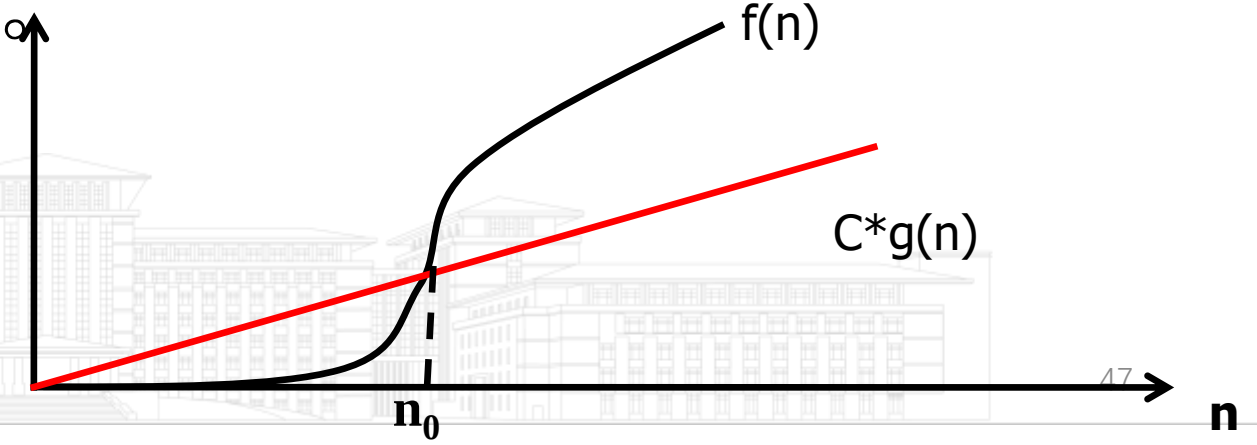


图2－1 时间复杂度函数曲线

定义2.2

如果存在两个正常数c和n_0，对于所有的$n \geq n_0$，有$|f(n)| \geq c|g(n)|$，则记作：$f(n)=\Omega(g(n))$。

- 当n充分大时，f(n)有下界，一个常数倍的g(n)是f(n)的一个下界。
- f(n)的阶不低于g(n)的阶。
- 在确定f(n)的下界时，总是试图求出最大的g(n)。
- 有关证明中，找出c和n_0是关键。



Ω性质

对于非负的f(n)和g(n)，根据定义2.2，有如下性质：

1. $\Omega(f(n))+ \Omega(g(n)) = \Omega(\min(f(n), g(n))$；
2. $\Omega(f(n))+ \Omega(g(n)) = \Omega(f(n)+g(n))$；
3. $\Omega(f(n))\ \Omega(g(n)) = \Omega(f(n)\ g(n))$；
4. 如果$g(n) = \Omega(f(n))$，则$\Omega(f(n))+ \Omega(g(n)) = \Omega(f(n))$；
5. $\Omega(cf(n)) = \Omega(f(n))$，其中$c$是一个正的常数；
6. $f(n) = \Omega(f(n))$。

定义2.3

如果存在正常数c_1,c_2和n_0，对于所有的$n \geq n_0$，有$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$，则记作：$f(n)= \Theta(g(n))$。

若$f(n) = \Theta(g(n))$，说明

- f(n)和g(n)时间复杂度相同，或者说二者是“同阶”的

渐近表示函数的若干性质

- 传递性
  - $f(n)=\Theta(g(n)),\quad g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$;
  - $f(n)=O(g(n)),\quad g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$;
  - $f(n)=\Omega(g(n)),\quad g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$;
- 反身性
  - $f(n)=\Theta(f(n))$;
  - $f(n)=O(f(n))$;
  - $f(n)=\Omega(f(n))$。

常用的整数求和公式

$$\sum_{1\le i\le n} 1 = \Theta(n)$$

$$\sum_{1\le i\le n} i = n(n+1)/2 = \Theta(n^2)$$

$$\sum_{1\le i\le n} i^2 = n(n+1)(2n+1)/6 = \Theta(n^3)$$

通式：$$\sum_{1\le i\le n} i^k = \frac{n^{k+1}}{k+1} + \frac{n^k}{2} + \text{低次项} = \Theta(n^{k+1})$$

一些数学证明方法

- 直接证明：P→Q
- 间接证明：
  - 反证法
  - 举反例
- 数学归纳法：
  - 初始归纳：i=1 结论成立；
  - 归纳假设：若i=n-1时成立；
  - 归纳证明：证明i=n时成立。

2.3 用SPARKS语言写算法

- 基本数据类型和变量命名规则
- 运算符
- 条件语句和循环语句
- 跳转语句
- 算法定义
- 算法的同质异相

基本数据类型和变量命名规则

- 基本数据类型：

  整型(integer)，实型(real)，布尔型(boolean)，字符型(char)

  布尔值：true，false

- 变量命名规则：

  以字母开头，不允许使用特殊字符，不允许与保留字重复

  例：integer x, y; char ch;

- 数组：任意整数下界和上界的多维数组：integer $A(l_1{:}u_1,\ldots,l_n{:}u_n)$

  例：integer A(1:10); real B(3:6 , 1:4);

运算符

- 赋值语句：变量 ← 表达式
- 逻辑运算符：and , or , not
- 关系运算符：< ，≤，=，≠，>，≥

条件语句和循环语句

- if语句

  if 条件 then S1
  endif

  if 条件 then S1
  else S2
  endif

- case语句

  case
  : 条件 1: S_1
  …
  : 条件 n: S_n
  : else : S_{n+1}
  endcase

- while语句

  while 条件 do
  S
  repeat //条件为假，循环结束

- loop语句

  loop//条件为真，循环结束
  S
  until 条件 repeat

- for语句

  for 变量←初值 to 终值 by 变量增值 do
  S
  repeat

跳转语句

- goto 标号
- exit
  - 转到含有exit的最内层循环语句后面的第一条语句
- cycle
  - 结束本次循环
- return (<表达式>)

算法定义

- SPARKS语言的输入、输出：
  - read（参数表）；print（参数表）；
- SPARKS语言的函数(过程)

```
procedure  NAME(<参数列表>)
 <说明部分>
      S
return(<表达式>)
end  NAME
```

函数名通常用大写字母

说明部分说明参数的数据类型
和函数中使用的变量
parameters　形式参数
global　全局变量
local　　局部变量

算法的同质异相

```
procedure  MAX(A, n, j)
 parameters  real  A(1:n);
 parameters  integer  n, j;
 global real  xmax;
 local  integer  i;
xmax←A(1);
for  i←2  to n do
    if  A(i)>xmax  then
          xmax←A(i);    j←i
    endif
repeat
return xmax
end  MAX
```

```
procedure  MAX(A, n, j)
global  xmax,   A(1:n);
int  i, n, j;
xmax←A(1);
for  i←2  to n do
    if  A(i)>xmax  then
          xmax←A(i);    j←i
    endif
repeat
return xmax
end  MAX
```

作业

1. 如果$g(n) = \Omega(f(n))$，则 $\Omega(f(n))+ \Omega(g(n)) = \Omega(f(n))$； ？
2. $\Omega(cf(n)) = \Omega(f(n))$，其中c是一个正的常数; ?
3. $\Theta(f(n))+ \Theta(g(n)) = \Theta(\min(f(n), g(n))$?
4. $\Theta(f(n))+ \Theta(g(n)) = \Theta(\max(f(n), g(n))$?
5. $\Theta(f(n))+\Theta(g(n)) = \Theta(f(n)+g(n))$?
6. $O(f(n)) * O(g(n)) = O(f(n) * g(n))$

- 若成立，证明之；不成立，举反例。

课程之外

- 授人以鱼不如授人以渔
- 开阔思路,训练思维
- 方法永远拥有理性的特点

本 章 结 束