第1章 算法概述

上海大学计算机工程与科学学院
岳晓冬

学习要点

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握复杂性的阶及表示
- 掌握用C++语言描述算法的方法。

1.1 算法与程序

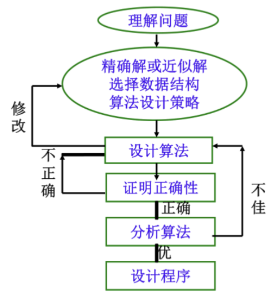
- 算法定义：

是一个有穷规则的集合。这些规则规定了解决某一问题的一个运算序列。

- 算法应该具有五个特性：

有限性、确定性、输入、输出、可行性。

问题求解过程



1.2 算法复杂性分析（1）

1. 计算机资源：时间、空间
2. 复杂性：所需资源多少
3. 算法复杂性：算法运行时所需资源的量
4. 算法复杂性分析目的：分析问题复杂性、算法是否可行，选择最好算法
5. 时间复杂性：所需时间资源的量$T(n)$
6. 空间复杂性：所需空间资源的量$S(n)$

- 其中n是问题的规模（输入大小）

算法复杂性分析（2）

- 时间复杂性细化--3种典型的复杂性：
  最坏、最好、平均复杂性
- 记T(I)为输入I时的问题的算法复杂性

几个复杂性参照函数

C
Log n
$\log^2 n$
n
n log n
n^2
n^3
2^n
n!

增长的阶

- 不同的计算过程在消耗计算资源的速率上可能存在着巨大差异，描述这种差异的一种方法是用**增长的阶**的记法，以便我们理解在输入变大时，某一计算过程所需资源的粗略度量情况。
- 令**n**是一个参数，它能作为问题规模的一种度量，令**f(n)**是一个计算过程在处理规模为**n**的问题时所需要的资源量。对某函数**g(n)**，称**f(n)**具有复杂性阶**Δ(g(n))**，记为 **f(n)= Δ(g(n))**，其中**Δ**是O、Ω、o、θ等。
- 含义见下面的几页。

若干符号及其意义：O(f)，Ω(f)，θ(f)，o(f)

- 在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。

（1）渐近上界记号O

$O(g(n))$ = { $f(n)$ | 存在正常数c和n_0使得对所有$n \geq n_0$有：

$0 \leq f(n) \leq cg(n)$ }

（2）渐近下界记号Ω

$\Omega(g(n))$ = { $f(n)$ | 存在正常数c和n_0使得对所有$n \geq n_0$有：

$0 \leq cg(n) \leq f(n)$ }

算法的五个特性

- 输入： 算法开始执行执行之前指定初始值（有零个或多个输入
- 输出：产生与输入相关的量（至少有一个）。
- 确定性：每一条规则都是明确、无二义的。
- 有限性：求解问题的运算序列，必须在有限的计算步后停止。
- 可行性：每一计算步都是基本的、可实现的。

计算序列与算法

1. 计算序列：

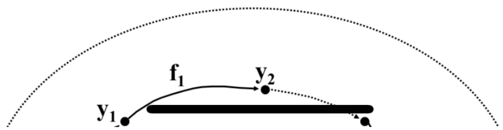
若对于**I** 的每一个输入**x**,由状态函数**f** 定义一个计算序列： y_0，y_1，y_2，……

其中：$y_0 = x$； $y_{k+1} = f(y_k)$ $(k \geq 0)$。

若一个计算序列在第**k**步终止，且**k**是使y_K 处于接受状态的最小整数，则称y_k是由**x**产生的输出。

2. 算法的另一种描述：

一个算法是对于**I** 中所有输入**x**, 都能在有穷步内终止的一个计算序列。



增长的阶

- 对于线性的过程，问题规模扩大一倍，资源增长一倍；
- 对于指数增长的过程，问题规模加1，资源就增大一个常数倍；
- 对于对数增长的过程，问题规模增大一倍，所需资源增长一个常数

各记号在等式和不等式中的意义

- $f(n)=\theta(g(n))$的确切意义是：$f(n) \in \theta(g(n))$。
- 一般情况下，等式和不等式中的渐近记号$\theta(g(n))$表示$\theta(g(n))$中的某个函数。
- 例如：$2n^2+3n+1=2n^2+\theta(n)$ 表示
- $2n^2+3n+1=2n^2+f(n)$，其中$f(n)$ 是$\theta(n)$中某个函数。
- 等式和不等式中渐近记号O,o和Ω的意义是类似的。

各记号的记忆性比较

- $f(n)=O(g(n))$ ≈ $f(n)$阶不超过$g(n)$阶
- $f(n)=\Omega(g(n))$ ≈ $f(n)$阶以$g(n)$阶为下界
- $f(n)=\theta(g(n))$ ≈ $f(n)$与$g(n)$等同
- $f(n)=o(g(n))$ ≈ $f(n)$阶小于$g(n)$阶

（3）高阶记号 o

$o(g(n)) = \{ f(n) \mid$ 对于任何正常数 $c>0$，存在正数和 $n_0>0$ 使得对所有 $n \geq n_0$ 有：$0 \leq f(n) < cg(n) \}$

等价于 $f(n) / g(n) \to 0$，as $n \to \infty$。

- $f(n)$ 的阶比 $g(n)$ 的阶低

（4）同阶记号 θ

$\theta(g(n)) = \{ f(n) \mid$ 存在正常数 c_1, c_2 和 n_0 使得对所有 $n \geq n_0$ 有：$c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

复杂性阶符号的理解举例

- 称f(n)具有θ(g(n))，记为 f(n)= θ(g(n))，如果存在与n无关的整数 c_1 和 c_2，使得：

  $c_1 g(n) \leq f(n) \leq c_2 g(n)$

  对于任何足够大的n都成立。

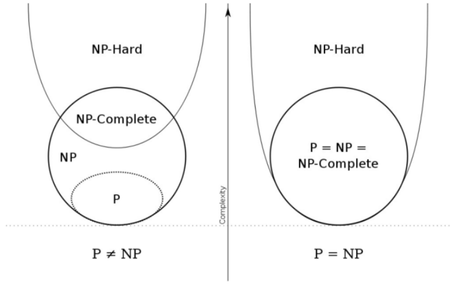
举例

求n！递归算法f(n)

f(n)=1，1，2，6，24，…，nf(n-1)，…

```
f(n){
   if(n==0 || n==1) return 1;
   return n*f(n-1);
}
```

求阶乘的递归算法时间和空间需求增长都是θ(n)；但对于迭代的算法，时间增长为Θ(n)，空间增长为θ(1)（常数）。



P: Polynomial，多项式时间可解的判定问题。
NP: Nondeterministic Polynomial，非确定性多项式时间可解的判定问题。可在多项式时间复杂度内对猜测进行验证。
NPC: NP-Complete，多项式时间内可转化为任意NP问题，可视为代表性NP问题。

P问题是确定计算模型下的易解问题，NP问题是非确定性计算模型下的易验证问题。

算法的时间复杂性

（1）最坏情况下的时间复杂性

$T_{max}(n) = \max\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$

（2）最好情况下的时间复杂性

$T_{min}(n) = \min\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$

（3）平均情况下的时间复杂性

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

其中I是问题的规模为n的实例，$p(\mathrm{I})$是实例I出现的概率。

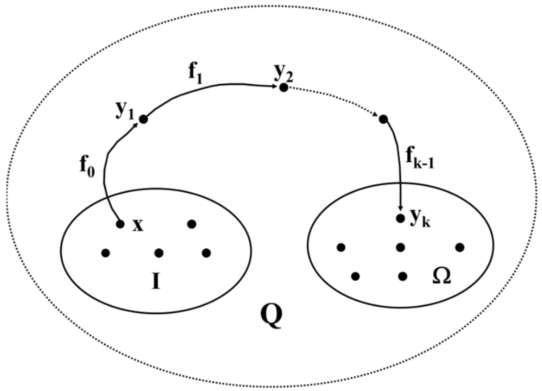
算法渐近复杂性

- 假定当 $n\to\infty$时, $T(n)\to\infty$;
- 取$t(n)$， 使$(T(n) - t(n))/ T(n)\to 0$， 当 $n\to\infty$;
- $t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。
- 在数学上， $t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。

状态的最小整数，则称y_k是由x产生的输出。

2. 算法的另一种描述：

一个算法是对于I 中所有输入x, 都能在有穷步内终止的一个计算序列。



程序

- 程序是算法用某种程序设计语言的具体体现
- 程序=算法＋数据结构（*Nicklaus Wirth*）
- 程序可以不满足算法的性质(4)有限性。
- 例如：操作系统，是一个在无限循环中执行的程序，因而不是一个算法。

问题求解过程

渐近分析记号的若干性质--传递性

（1）传递性：

$f(n)=\theta(g(n))$， $g(n)=\theta(h(n)) \Rightarrow f(n)=\theta(h(n))$；

$f(n)=O(g(n))$， $g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；

$f(n)=\Omega(g(n))$， $g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；

$f(n)=o(g(n))$， $g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$；

渐近分析记号的若干性质

--反身性、对称性、互对称性

（2）反身性：

$f(n)=\theta(f(n))$；

$f(n)=O(f(n))$；

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

（3）对称性：

$f(n)=\theta(g(n)) \Leftrightarrow g(n)=\theta(f(n))$.

（4）互对称性：

$f(n)=O(g(n)) \Leftrightarrow g(n)=\Omega(f(n))$ ；

渐近分析记号的若干性质

--算术运算

- （5）算术运算：
- $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$ ；
- $O(f(n))+O(g(n)) = O(f(n)+g(n))$ ；
- $O(f(n))*O(g(n)) = O(f(n)*g(n))$ ；

渐近分析记号的若干性质
--算术运算

- （5）算术运算：
- $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$；
- $O(f(n))+O(g(n)) = O(f(n)+g(n))$；
- $O(f(n))*O(g(n)) = O(f(n)*g(n))$；
- $O(cf(n)) = O(f(n))$；C是正常数
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(n))$。

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

对于任意$f_1(n) \in O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n\geq n_1$，有$f_1(n) \leq c_1 f(n)$。

类似地，对于任意$g_1(n) \in O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n\geq n_2$，有$g_1(n) \leq c_2 g(n)$。

令$c_3=\max\{c_1, c_2\}$，$n_3=\max\{n_1, n_2\}$，$h(n)= \max\{f(n),g(n)\}$。

则对所有的 $n \geq n_3$，有

$$
\begin{aligned}
f_1(n) + g_1(n) &\leq c_1 f(n) + c_2 g(n) \\
&\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n)) \\
&\leq c_3 2 \max\{f(n),g(n)\} \\
&= 2c_3 h(n) = O(\max\{f(n),g(n)\}).
\end{aligned}
$$

算法渐近复杂性分析中常用函数

（1）取整函数

$\lfloor x \rfloor$：不大于x的最大整数；

$\lceil x \rceil$：不小于x的最小整数。

取整函数的若干性质

1. $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$；
2. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
3. 对于$n \geq 0$，$a,b>0$，有：

   $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
4. $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
5. $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。

多项式函数

- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;
- $p(n) = \theta(n^d)$;
- $f(n) = O(1) \Leftrightarrow f(n) \le c$;
- $k \ge d \Rightarrow p(n) = O(n^k)$;
- $k \le d \Rightarrow p(n) = \Omega(n^k)$;
- $k > d \Rightarrow p(n) = o(n^k)$;

指数函数、对数函数

$a>1 \Rightarrow \quad \lim\limits_{n\to\infty}\frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$

记号： $\log n = \log_2 n$; $\lg n = \log_{10} n$; $\ln n = \log_e n$;

$|x| \le 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - 6$.

for $x > -1$, $\frac{x}{1+x} \le \ln(1+x) \le x$

对任何 $a > 0$, $\lim\limits_{n\to\infty}\frac{\log_b n}{n^a} = 0 \quad \Rightarrow \log_b n = o(n^a)$

阶乘函数

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

$$n! = 1 \cdot 2 \cdot 36 \;\; n$$

Stirling公式

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

算法分析的基本法则

- 非递归算法：

（1）for / while 循环

循环体内计算时间*循环次数；

（2）嵌套循环

循环体内计算时间*所有循环次数；

（3）顺序语句

各语句计算时间相加；

（4）if-else语句

if语句计算时间和else语句计算时间的较大者。

求幂-传统方法

$b^n = b*b^{n-1}$

$b^0 = 1$

```
pow(b,n){
  if(n==0) return 1;
  return b*pow(b,n-1);
}
```

时间和空间增长θ(n)

求幂-二分

$b^n = (b^{n/2})^2$ n是偶数

$b^n = b*b^{n-1}$ n是奇数

```
fast_pow(b,n){
   if(n==0) return 1;
   if(n%2==0) square(fast_pow(b,n/2));
   else b*fast_pow(b,n-1);
}
```

时空增长的阶为 θ (log n)