算法设计与分析

第 1 章 绪论

金英　编写

计算机科学技术学院
软件学院

2021 年 2 月

目　录

第 1 章 绪论

1.1 算法概述

通俗地讲，算法是指解决问题的一种方法或一个过程。

严格地讲，**算法**是由若干条指令组成的有穷序列,且满足下述性质：

(1) 输入: 有零个或多个由外部提供的量作为算法的输入。

(2) 输出: 算法产生至少一个量作为输出。

(3) 确定性: 组成算法的每条指令是清晰的，无歧义的。

(4) 有限性: 算法中每条指令执行次数是有限的，执行每条指令的时间也是有限的。

算法与程序的区别：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)。例如: 操作系统，它是一个在无限循环中执行的程序，因而不是算法。可把操作系统的各种任务看成是一些单独的问题，每个问题由操作系统中的一个子程序通过特定的算法来实现。

描述算法可以有多种形式，如自然语言、表格、流程图、伪代码等。本课程将用C/C++伪代码描述算法。

1.2 算法复杂性分析

一个算法的复杂性的高低体现在运行该算法所需的计算机资源（主要指时间和空间资源）的多少上。算法的复杂性分为：时间复杂性和空间复杂性。

随着计算机要解决的问题越来越复杂，规模越来越大，对求解这类问题的算法作复杂性分析具有特别重要的意义。随着计算机技术的迅速发展，对空间的要求往往不如对时间的要求那样强烈。因此我们这里的分析主要强调时间复杂性的分析。

考虑时间复杂性的理由：

某些用户需要提供程序运行时间的上限（用户可接受的）；

所开发的程序需要提供一个满意的实时反应。

本课程考虑如下三种情况下的时间复杂度：

最坏情况；（重点讨论）；最好情况；平均情况。

- **时间复杂性的计算方法** （即估算运行时间的方法）：

加、减、乘、除、比较、赋值等操作，一般被看作是基本操作，并约定所用的时间

都是一个单位时间；通过计算这些操作分别执行了多少次来确定程序总的执行步数。一般地，一些关键操作执行的次数决定了算法的时间复杂度。

例 1：　二分查找算法。

```
int bsearch(K,L,H)
{
    if (H<L)    return(-1);                          2
    else
    {
        mid=(L+H) /2;                                3
        element=A[mid];                              2
        if (element == K)                            1
            return(mid)；                            1
        else if （A[mid]>K）                         2
            return(bsearch(K,L, mid-1)) ;            3+T(n/2)
        else return(bsearch(K, mid+1,H));            3+T(n/2)
       }
}
```

算法时间复杂性分析：

∵ T(n)=2　　当 n=0

T(n)=11+T(n/2)　　当 n≥1

∴　T(n) = O(*log n)*

例 2：　寻找最大元素。

```
int Max(T a[], int    n)
{    //寻找 a[0:n-1]中的最大元素
     int pos = 0;
     for (int i =1; i < n; i++)
          if (a[pos] < a[i])
               pos = i;
     return pos;
}
```

算法时间复杂性分析：

这里的关键操作是比较。for 循环中共进行了 n-1 次比较。Max 还执行了其它比较，如 for 循环每次比较之前都要比较一下 i 和 n。此外，Max 还进行了其他的操作，如初始化 pos、循环控制变量 i 的增量。这些一般都不包含在估算中，若纳入计数，则操作计数将增加一个常量。

∴T(n)= O(n)

- **渐进复杂性**（Asymptotic Complexity）

确定程序的操作计数和执行步数的目的是为了比较两个完成同一功能的程序的时间复杂性，预测程序的运行时间随着问题规模变化的变化量。设 T(n)是算法 A 的复杂性函数。一般说来，当 n 单调增加趋于∞时，T(n)也将单调增加趋于∞。如果存在函数$\tilde{T}(n)$，使得当$n\to\infty$时有$(T(n)-\tilde{T}(n))/T(n)\to 0$，则称$\tilde{T}(n)$是 T(n)当$n\to\infty$时的渐近性态，或称$\tilde{T}(n)$是算法 A 当$n\to\infty$的渐近复杂性（而与 T(n)相区别）。

因为在数学上，$\tilde{T}(n)$是 T(n)当$n\to\infty$时的渐近表达式，$\tilde{T}(n)$是 T(n)中略去低阶项所留下的主项，所以它无疑比 T(n)来得简单。

例如，　当 $T(n)=3n^2+4n\log n+7$时，

$$\tilde{T}(n)=3n^2$$

要比较两个算法的渐近复杂性的阶是否相同时，只要确定出各自的阶，就可以知哪一个算法的效率高。因此，渐近复杂性分析只要关心$\tilde{T}(n)$的阶就够了，不必关心包含在$\tilde{T}(n)$中的常数因子。所以，我们可以进一步简化$\tilde{T}(n)$如下:

$$\tilde{T}(n)=n^2$$

综上分析，我们已经给出了简化算法复杂性分析的方法和步骤，即只考虑当问题的规模充分大时，算法复杂性在渐近意义下的阶。为此引入渐近符号，在引入之前，首先给出常用的渐近函数，如表 1-1 所示。

表 1-1 常用的渐近函数

函数	名称	函数	名称
1	常数	n^2	平方
log n	对数	n^3	立方
n	线性	2^n	指数
n log n	n 倍 log n	n!	阶乘

在下面的讨论中，用 f(n)表示一个程序的时间或空间复杂性，它是问题规模 n（一般是输入规模）的函数。由于一个程序的时间或空间需求是一个非负的实数，我们假定函数 f(n)对于 n 的所有取值均为非负实数，而且还可假定 n≥0。

- **渐进记号**

(1) **渐近记号 O 的定义**

f(n)=O(g(n))当且仅当存在正的常数 C 和 n_0，使得对于所有的 $n \geq n_0$，有 $f(n) \leq Cg(n)$。此时，称 g(n)是 f(n)当 n 充分大时的一个上界。

例：

$3n+2 = O(n)$　　可取 C=4，$n_0=2$

$100n+6 = O(n)$　　可取 C=101，$n_0=6$

$10n^2+4n+3 = O(n^2)$　　可取 C=17，$n_0=3$（当取 C=17 时，$n_0=1$ 即可）

$6\times2^n+n^2 = O(2^n)$　　可取 C=7，$n_0=4$

三点注意事项：

① 用来作比较的函数 g(n)应该尽量接近所考虑的函数 f(n)。

如：$3n+2=O(n^2)$ 松散的界限；

$3n+2=O(n)$ 较好的界限。

② 不要产生错误界限。

如：$n^2+100n+6$，当 $n<8$ 时，$n^2+100n+6<108n$

由此就认为 $n^2+100n+6 = O(n)$

事实上，对任何正的常数 C，只要 $n>C-100$ 就有 $n^2+100n+6>C\times n$。

同理，$3n^2+4\times2^n=O(n^2)$是错误的界限。

③ f(n)=O(g(n))不能写成 g(n)=O(f(n))，因为两者并不等价。

(2) 渐近记号Ω的定义

$f(n)=\Omega(g(n))$ 当且仅当存在正的常数 C 和 n_0，使得对于所有的 $n \geq n_0$，有 $f(n) \geq Cg(n)$。此时，称 g(n)是 f(n)当 n 充分大时的一个下界。

(3) 渐近记号 Θ 的定义

$f(n)= \Theta(g(n))$当且仅当存在正的常数 C_1，C_2 和 n_0，使得对于所有的 $n \geq n_0$，有 $C_1g(n) \leq f(n) \leq C_2g(n)$。此时，称 f(n)与 g(n)同阶。

例：　$3n+2 = \Theta(n)$

$10n^2+4n+2=\Theta(n^2)$

$5\times2^n+ n^2=\Theta(2^n)$

(4) 渐进记号小 o 定义

f(n)=o(g(n))当且仅当 f(n)=O(g(n))和 g(n) ≠O(f(n))。

此时，g(n)是 f(n)的一个绝对上界。

例： $4n\log n+7 = o(n^2)$

(5) 渐进记号小 ω 定义

f(n)=ω(g(n))当且仅当 f(n)= Ω(g(n))和 g(n) ≠Ω(f(n))。

此时，g(n)是 f(n)的一个绝对下界。

1.3 递归方程解的渐进阶求法

本节将介绍三种求递归方程解的渐近阶的方法，即找出解的渐进“Θ”或“O”界的方法。

1.3.1 套用公式法（Master method）

套用公式法给出求解如下形式的递归式的方法：

$$T(n) = aT(n/b) + f(n) \quad (1)$$

其中的 a≥1 和 b>1 是常数，f(n)是一个确定的渐进正函数。

式(1)是一类分治法的时间复杂性所满足的递归关系，即一个规模为 n 的问题被分为规模均为 n/b 的 a 个子问题，递归地求解这 a 个子问题，然后通过对这 a 个子问题的解的综合，得到原问题的解。如果用 T(n)表示规模为 n 的原问题的复杂性，用 f(n)表示把原问题分成 a 个子问题和将 a 个子问题的解综合为原问题的解所需的时间，我们便有方程式(1)。

递归方程解的渐近阶提供三个可套用的公式，这个方法依据的是如下定理。

定理 1： 设 a≥1 和 b >1 是常数，f(n)是定义在非负整数上的一个确定的非负函数。又设 T(n)也是定义在非负整数上的一个非负函数，且满足递归方程式(1)。方程式(1)中的 n/b 可以是$\lfloor n/b \rfloor$，也可以是$\lceil n/b \rceil$。那么在 f(n)的三类情况下，我们可有 T(n)的渐近估计式：

(1) 若对于某常数 ε>0，有 $f(n)=O(n^{\log_b a-\varepsilon})$，则 $T(n)=\Theta(n^{\log_b a})$

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

(3) 若对于某常数 ε>0，有 $f(n)=\Omega(n^{\log_b^a+\varepsilon})$，且对于某常数 c<1 和所有充分大的正整数 n 有 $af(n/b)\le cf(n)$，则 $T(n)=\Theta(f(n))$。

这里涉及的三类情况，都是将 f(n)与 $n^{\log_b^a}$ 作比较。定理直观地告诉我们，递归方程的渐近阶由这两个函数中的较大者决定。

- 在第一类情况下，$n^{\log_b^a}$ 较大，则 $T(n)=\Theta(n^{\log_b^a})$。
- 在第二类情况下，两个函数一样大，则 $T(n)=\Theta(n^{\log_b^a}\log n)=\Theta(f(n)\log n)$。（即以 n 的对数作为因子乘上 f(n)与 T(n)同阶）
- 在第三类情况下，f(n)较大，则 $T(n)=\Theta(f(n))$。

例 1： T(n)=9*T*(n/3)+n

此时，a=9，b=3，f(n)=n，$\therefore n^{\log_b a}=n^{\log_3 9}=n^2$，取 $\varepsilon\in(0,1]$，便有 $f(n)=O(n^{\log_b^a-\varepsilon})$，可套用第一类情况得 $T(n)=\Theta(n^2)$。

例 2： T(n)=*T*(2n/3)+1

此时，a=1，b=3/2，f(n)=1，$\therefore n^{\log_b a}=n^{\log_{3/2}1}=1=f(n)$，可套用第二类情况得 $T(n)=\Theta(n^0\log n)=\Theta(\log n)$。

例 3： T(n)=3*T*(n/4)+*n*log*n*

此时，a=3，b=4，f(n)=*n*log*n*，$\therefore n^{\log_b^a}=n^{\log_4^3}=O(n^{0.793})$，取 $\varepsilon\approx 0.2$，便有 $f(n)=\Omega(n^{\log_b^a+\varepsilon})$。并且，$af(n/b)=3(n/4)\log(n/4)\le(3/4)n\log n=Cf(n)$，即取 C=3/4<1，对充分大的 n 都有 $af(n/b)\le cf(n)$。可套用第三类情况得 $T(n)=\Theta(n\log n)$。

例 4： T(n)=2T(n/2)+nlogn（本方法对之无能为力的例子）

此时，a=2，b=2，f(n)=nlogn，$\therefore n^{\log_b^a}=n$，虽然 f(n)渐近地大于 $n^{\log_b^a}$，但 f(n)并不是多项式地大于 $n^{\log_b^a}$。因为对于任意的正常数 ε，$f(n)/n^{\log_b^a+\varepsilon}=n\log n/n^{1+\varepsilon}=n^{-\varepsilon}\log n\to 0$，即 f(n)在第二类情况和第三类情况的间隙里，本方法对它无能为力。

注意：定理中的一些细节不能忽视：

在第一类情况下，f(n)不仅必须比 $n^{\log_b a}$ 小，而且必须是多项式地比 $n^{\log_b a}$ 小，即 f(n)必须渐近地小于 $n^{\log_b a}$ 与 $n^{-\varepsilon}$ 的积，ε 是个正的常数。

在第三类情况下，f(n)不仅必须比 $n^{\log_b a}$ 大，而且必须是多项式地比 $n^{\log_b a}$ 大，即 f(n)必须渐近地大于 $n^{\log_b a}$ 与 $n^{+\varepsilon}$ 的积，ε 是个正的常数。还要满足附加的“正规性”条件：$af(n/b)\le cf(n)$。这个附加的“正规性”条件的直观含义是 a 个子问题的再分解和再综合所需的时间最多与原问题的分解和综合所需的时间同阶。我们将碰到的以多项式为界的函数基本上都满足这个正规性条件。

1.3.2 代入法*(Substitution Method)

代入法是先猜某个界存在，然后用数学归纳法证明该界的正确性。用这个办法既可估计上界也可估计下界。这个方法的关键步骤如下：

① 预先对解答做出推测；

② 用数学归纳法证明推测的正确性。

例 1： 要估计 T(n)的上界。假设 T(n)满足下述递归方程：

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

推测 T(n)=O(nlog n)

即：推测存在常数 C>0 和自然数 n_0，使得当 $n \ge n_0$ 时有：

$$T(n) \le Cn\log n \qquad (3)$$

下面来证明当 $n \ge n_0$ 时，$T(n) \le Cn\log n$，其中 C>0 是某个常数，$n \ge n_0$ 是某个自然数。

T(1)=1，C·1·log1=0，1 不小于 0，即当 n=1 时不成立。但是

T(2)=4=2·2·log2≤C·2log2，即取 C 为不小于 2 的常数，当 n=2 时成立。

T(3)=5≤C·3log3，即取与 n=2 时相同的常数 C，当 n=3 时成立。

即当 $2 \le n \le 3 < 4$，亦即当 $2^1 \le n < 2^2$ 时，$T(n) \le Cn\log n$。

假设当 $2^{k-1} \le n < 2^k (k \ge 2)$ 时，有 $T(n) \le Cn\log n$。那么，当 $2^k \le n < 2^{k+1}$ 时，$2^{k-1} \le \lfloor n/2 \rfloor < 2^k$，根据前面的假设就有 $T(\lfloor n/2 \rfloor) \le C\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor$。

因为$\lfloor n/2 \rfloor \le n/2$，$\log\lfloor n/2 \rfloor \le lon\frac{n}{2}$，所以$T(\lfloor n/2 \rfloor) \le C\frac{n}{2}\log\frac{n}{2}$。于是

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\le 2 \cdot C\frac{n}{2}\log\frac{n}{2} + n \\ &= Cn(\log n - 1) + n \\ &= Cn\log n - (C-1)n \\ &\le Cn\log n \end{aligned}$$

即：取 C 为不小于 2 的常数和自然数 n_0=2，对于所有的 $n \ge n_0$，有$T(n) \le Cn\log n$。这样证明了推测是正确的，所以递归方程式(2)的解的渐近阶是 O(*n*log*n*)。

1.3.3 迭代法*(Iteration Method)

用这个方法估计递归方程解的渐近阶不要求推测解的渐近表达式，但要求较多的代数运算技巧。

例 1：考虑递归方程：

$$T(n) = 3T(\lfloor n/4 \rfloor) + n \qquad (4)$$

接连迭代 2 次可将右端展开为：

$$\begin{aligned} T(n) &= n + 3T(\lfloor n/4 \rfloor) \\ &= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor \lfloor n/4 \rfloor /4 \rfloor)) \\ &= n + 3(\lfloor n/4 \rfloor + 3(\lfloor \lfloor n/4 \rfloor /4 \rfloor + 3T(\lfloor \lfloor \lfloor n/4 \rfloor /4 \rfloor /4 \rfloor))) \end{aligned}$$

$\because \lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/(ab) \rfloor$

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

迭代 i 次后，将有：

$$\begin{aligned} T(n) = n + 3(&\lfloor n/4 \rfloor + 3(\lfloor n/4^2 \rfloor + 3(\lfloor n/4^3 \rfloor + \\ &\Lambda + 3(\lfloor n/4^i \rfloor + 3T(\lfloor n/4^{i+1} \rfloor))\Lambda))) \end{aligned} \qquad (5)$$

当$\lfloor n/4^{i+1} \rfloor = 0$时，式(5)不再是递归方程。这时

$$T(n) = n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/4^2 \rfloor + 3(\lfloor n/4^3 \rfloor + \Lambda + 3(\lfloor n/4^i \rfloor + 3T(0))\Lambda)))$$

$$T(n) \le n+3(n/4+3(n/4^2+3(n/4^3+ \ldots +3(n/4^i+3T(0))\ldots)))$$

$$= n + \frac{3}{4}n + \frac{3^2}{4^2}n + \frac{3^3}{4^3}n + \Lambda + \frac{3^i}{4^i}n + 3^{i+1}T(0)$$

$$= (1 + \frac{3}{4} + \frac{3^2}{4^2} + \frac{3^3}{4^3} + \Lambda + \frac{3^i}{4^i})n + 3^{i+1}T(0)$$

$$= 4\left(1 - \left(\frac{3}{4}\right)^{i+1}\right)n + 3^{i+1}T(0)$$

$$< 4n + 3^{i+1}T(0) \qquad (6)$$

由于$\lfloor n/4^i \rfloor \geq \lfloor n/4^{i+1} \rfloor = 0$，$\therefore n \geq 4^i$，$\therefore i \leq \log_4 n$，从而

$3^{i+1} = 3 \times 3^i \leq 3 \times 3^{\log_4 n} = 3n^{\log_4 3}$

代入(6)式得：　$T(n) < 4n + 3n^{\log_4 3}T(0)$

$\therefore$　$T(n) = O(n)$

本章习题

1. 按照渐近阶从低到高的顺序排列以下表达式：
   $n!, 4n^2, \log n, 3^n, 20n, 2, n^{2/3}$

2. 对下列各组函数 f (n) 和 g (n)，确定 f (n) = O (g (n)) 或 f (n) =Ω(g (n))或 f(n) =θ(g(n))，并简述理由。

   (1) f(n) =log n^2　　g (n) = log n+5

   (2) f(n) = log n^2　　g (n) = $\sqrt{n}$

   (3) f(n) = n　　g(n)= $\log^2$ n

   (4) f(n) =nlog n　　g (n) = log n

   (5) f(n)=10　　g (n) = log 10

   (6) f(n) = $\log^2$ n　　g (n) = log n

   (7) f(n)=2^n　　g (n) =100 n^2

   (8) f(n)=2^n　　g(n)= 3^n

3. 对于任意常数正整数 k，试按照函数阶从低到高的次序排列函数
   n^k，　$\log n^k$，　$n^{1/k}$，　$(\log n)^k$，　$(1/k)^n$。

4. 用定义证明下列结论：

   (1)　$3n^2 + 5n + 2 = O(n^2)$

   (2)　3n+7=Ω(n)

(3) 设 f(n)与 g(n)都是渐近非负函数。利用 θ 记号的基本定义来证明：
max(f(n),g(n))= θ(f(n)+g(n))。

(4) $n^2/2-3n=\theta(n^2)$

(5) $6n^3 \neq \theta(n^2)$.

(6) $f(n)=\theta(g(n))$ **iff** $f(n)=O(g(n))$且 $f(n)=\Omega(g(n))$

(7) $2n^2 \neq o(n^2)$

(8) $2n=o(n^2)$

5. 求下列递归方程解的渐进阶。

(1) $T(n)=2T(n/4)+n^{1/2}$.

(2) $T(n) = 2T(n^{1/2}) + n$.

参考书

1. 王晓东. 计算机算法设计与分析 (第 5 版)[M]. 北京:电子工业出版社，2018.
2. Thomas H.Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms (Second Edition)[M]. The MIT Press, 2013.