算法导论

范潇

2024　春

目录

第一章　算法分析

当我们在研究算法时，需要分析算法是否正确以及是否高效，以便于挑选出最优的算法加以使用。

一个正确的算法需要保证对于所有规模的输入都能给出正确的输出。因此，在证明算法的正确性时，我们往往使用数学归纳法。

一个高效的算法需要在输入规模较大时仍能较为快速地给出输出，为此，我们使用渐进分析来描述算法的效率。

1.1. RAM 模型

为了简化分析算法的过程，我们要对运行算法的机器进行建模以及简化。我们所采用的模型被称为 RAM 模型，即“Random Access Model”，其中的核心假设如下：

1. 内存 (memory) 由一系列字 (word) 组成，可以像数组编号，并通过编号访问
2. CPU 一次只能取出一个字，并对其进行操作
3. 获取一个字只需要一个基本操作。因此我们假设一个字所含的比特数 $w = c\lg n$，其中 n 为数据规模，这保证了其能够在一个字内处理所需数据的编号。

1.2. 正确性分析

1.2.1. 循环不变量

通常一个算法中有一个或多个循环语句，并且某些关键性质通过这些循环进行维护，从开始到结束保持着某种不变量。这种不变量我们称其为“循环不变量”。

给定一个核心为循环的算法，首先我们要找出其中的“循环不变量”，然后进行归纳证明。归纳证明主要分为 3 步：

1. **初始状态：**循环不变量是否在进入第一轮循环之前正确
2. **维护：**如果循环不变量在某一轮循环之前正确，它是否在进入下一轮循环之前保持正确
3. **结束状态：**当循环结束时，该循环不变量能否提供有用的性质以便于说明该算法的正确性

对于 `for` 语句而言，这三部的时机分别位于：1）计数器赋初值之后（严格而言并且是在判断循环是否结束之前）；2）执行循环体然后计数器更新这一过程；3）计数器更新导致循环结束后。也就是说，对于 `for` 语句，计数器初始化代表着整个循环语句的初始化，但是和任何一轮循环无关；一轮循环依次包括：检查循环是否结束、执行循环体、更新计数器。

1.3. 渐进分析

在本节中讨论的函数的定义域都是正整数，并且当自变量足够大时，函数值为正。

Definition 1.3.1 (渐进符号).

$$\begin{aligned}
O(g(n)) &= \{f(n) : \exists c, n_0 > 0 \quad s.t. \quad \forall n \geq n_0 \quad f(n)/g(n) \leq c\} \\
\Omega(g(n)) &= \{f(n) : \exists c, n_0 > 0 \quad s.t. \quad \forall n \geq n_0 \quad f(n)/g(n) \geq c\} \\
\Theta(g(n)) &= \{f(n) : \exists c_1, c_2, n_0 > 0 \quad s.t. \quad \forall n \geq n_0 \quad c_1 \leq f(n)/g(n) \leq c_2\} \\
o(g(n)) &= \{f(n) : \lim_{n\to\infty} f(n)/g(n) = 0\} \\
\omega(g(n)) &= \{f(n) : \lim_{n\to\infty} f(n)/g(n) = \infty\}
\end{aligned}$$

Remark.

记忆渐进符号中 f, g 的相对关系可以进行如下理解：渐进符号是一个集合，有许多函数，而该渐进符号却用一个函数来代表这个集合，那么这个函数是有特殊意义的，也就是各个渐进符号的含义。例如 $O(f)$ 就是一个函数的集合，其中的函数的一个紧上界为 f。

$O(g(n)), o(g(n))$ 中的函数的上界均可以用 $g(n)$ 来描述，区别在于前者的上界是“紧”的，而后者是“松”的——给出的上界远远超过实际值。类似地，$\Omega(g(n)), \omega(g(n))$ 中的函数的下界均可以用 $g(n)$ 来描述，区别在于前者的下界是“紧”的，而后者是“松”的——给出的下界远远小于实际值。而 $\Theta(g(n))$ 中的函数可以由 $g(n)$ 给出一个“紧”的估计。

渐近符号 $O, \Omega, \Theta, o, \omega$ 可以分别比作 $\leq, \geq, =, <, >$，享有着对应符号的性质，例如传递性，对称性等。但是，渐进符号不具有三歧性。

1.4. 分治法

分治法是设计算法的常用方法之一，它主要分为三步：分割、解决以及合并。它的主要思想便是将一个规模较大的问题拆分成多个规模较小的问题，然后分别解决这些较小的问题，最后将这些较小问题的解答合并，从而得到原先问题的解答。分治法通常涉及到递归，它的时间复杂度的一般形式为

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

这表示：1）我们假设当问题规模降低到一定程度时，便可以在常数时间内解决；2）当问题规模较大时，我们将原先规模为 n 的问题拆分为 a 个规模为 n/b 的子问题，将它们分别解决后然后进行合并，其中 $D(n), C(n)$ 分别表示拆分和合并所需要的时间，我们也可以用 $f(n)$ 来表示它们之和。

Remark.

一般而言，每个阶段我们将拆分得到多个规模相同的，且结构和原问题一致的子问题。如果我们需要拆分出来结构和原问题发生较大变化的子问题，我们则将它对应的时间复杂度算入合并所需的时间中。

这也就意味着，分析使用分治法的算法的时间复杂度时，需要求解一个递推函数。为此，本节将介绍三种求解方法。

1.4.1. 替换法

替换法的核心步骤为

1. 猜测解的形式
2. 使用数学归纳法进行证明

第一步通常需要较多的经验。当然也可以通过不断缩小上下界来找到紧的界。

在进行数学归纳法的证明时，我们可能会在证明初始情况遇到困难，此时我们可以将初始情况进行适当调整，因为我们进行的是渐进分析。在证明递推情况时，我们可能也会遇到问题，此时的解决方法通常是在原先估计的基础上，减去一个低阶量。

> *Remark.*
> 通过减去低阶量，往往能够使得我们的归纳假设变得更强，从而更轻松地完成证明。