第一章算法效率分析基础

整理者：Yu
资料整理自互联网，仅供学习用途

2017 年 3 月 5 日

同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。算法分析的目的在于选择合适算法和改进算法。一个算法的评价主要从时间复杂度和空间复杂度来考虑。随着硬件技术的不断发展和价格走低，空间复杂度在大部分情况下已经不是需要重点关注的问题了，然而时间效率的重要性并没有减弱到这种程度。本章将主要介绍时间复杂度。

为什么要进行算法分析?

- 预测算法所需要的资源
  - 计算时间（CPU 消耗）
  - 内存空间（RAM 消耗）
  - 通信时间（带宽消耗）
- 预测算法的运行时间
  - 在给定输入规模时，所执行的基本操作数量。
  - 或者称为算法复杂度（Algorithm Complexity）

如何衡量算法复杂度?

- 内存（Memory）
- 时间（Time）
- 指令的数量（Number of Steps）
- 特定操作的数量
- 磁盘访问数量
- 网络包数量
- 渐进复杂度（Asymptotic Complexity）

算法的运行时间与什么相关?

- 取决于输入的数据的初始状态。（例如：如果数据已经是排好序的，时间消耗可能会减少。）
- 取决于输入数据的规模。（例如：10个数排序和10^8个数排序）
- 取决于运行时间的上限。（因为运行时间的上限是对使用者的承诺。）

算法分析的种类：

- 最坏情况（Worst Case）：任意输入规模的最大运行时间。（Usually）
- 平均情况（Average Case）：任意输入规模的期待运行时间。（Sometimes）
- 最佳情况（Best Case）：通常最佳情况不会出现。（Bogus）

例如，在一个长度为n 的列表中顺序搜索指定的值，则

(1) 最坏情况：n 次比较

(2) 平均情况：n/2 次比较

(3) 最佳情况：1 次比较

而实际中，我们一般仅考量算法在最坏情况下的运行情况，也就是对于规模为n 的任何输入，算法的最长运行时间。这样做的理由是：

(1) 一个算法的最坏情况运行时间是在任何输入下运行时间的一个上界（Upper Bound）。

(2) 对于某些算法，最坏情况出现的较为频繁。

(3) 大体上看，平均情况通常与最坏情况一样差。

算法分析要保持大局观（Big Idea），其基本思路：

(1) 忽略掉那些依赖于机器的常量。

(2) 关注运行时间的增长趋势。

比如：$T(n) = 73n^3 + 29n^2 + 8888$ 的趋势就相当于T(n) = $\Theta(n^3)$。
渐近记号（Asymptotic Notation）通常有O、Θ 和Ω 记号法。Θ 记号渐进地给出了一个函数的上界和下界，当只有渐近上界时使用O 记号，当只有渐近下界时使用Ω 记号。尽管技术上Θ 记号较为准确，但通常仍然使用O 记号表示。例如,线性复杂度$O(n)$ 表示每个元素都要被处理一次。平方复杂度$O(n^2)$ 表示每个元素都要被处理n 次。

分析非递归算法的效率的通用方案

(1) 决定用哪个(哪些)参数作为输入规模的度量

(2) 找出算法的基本操作（作为一规律，它总是位于算法的最内层循环中）。

(3) 检查基本操作的执行次数是否只依赖输入规模。如果它还依赖一些其他的特性，则最差效率、平均效率以及最优效率（如果必要）需要分别研究。

表 1: 渐进记号表示法

Notation	Intuition	Informal Definition
$f(n) \in O(g(n)$	f is bounded above by g asymptotically	$\exists n > n_0, \lvert f(n)\rvert \le g(n) \cdot k$
$f(n) \in \Omega(g(n)$	f is bounded below by g asymptotically	$\exists n > n_0, \lvert f(n)\rvert \ge g(n) \cdot k$
$f(n) \in \Theta(g(n)$	f is bounded above and below by g asymptotically	$\exists n > n_0, g(n) \cdot k_1 \le f(n) \le g(n) \cdot k_2$

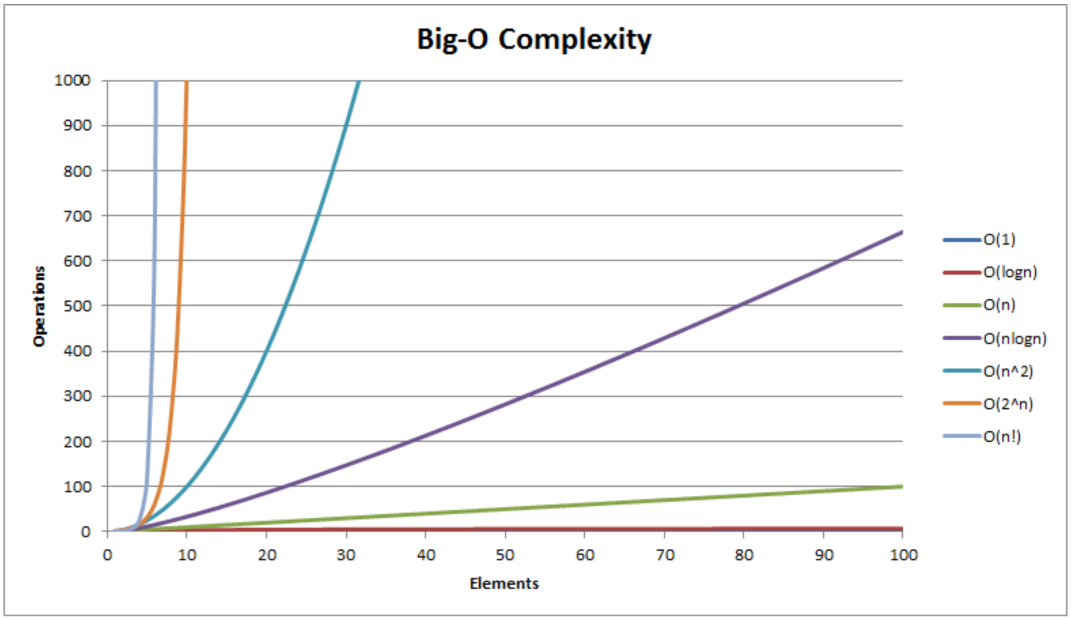

图 1: 基本的渐进效率图像

(4) 建立一个算法基本操作执行次数的求和表达式。

(5) 利用求和运算的标公式和法则来建立一个操作次数的闭合公式，或者至少确定它的增长次数。

```
def FindMaxElement(array):
    max = array[0]
    for i in range(len(array):
        if array[i] > max:
            max = array[i]
```

以以上代码为例，代码中最基本的执行语句为比较array与max的大小，其执行数量约为$n*(n-1)/2$，所以算法复杂度为$O(n^2)$。

表 2: 基本的渐进效率类型

复杂度	标记符号
常量（Constant）	$O(1)$
对数（Logarithmic）	$O(\log_2 n)$
线性（Linear）	$O(n)$
平方（Quadratic）	$O(n^2)$
立方（Cubic）	$O(n^3)$
指数（Exponential）	$O(2^n), O(k^n), O(n!)$