HZ BOOKS
华章IT

涵盖世界知名IT公司技术面试的程序设计问题及其解题思路
供不同复杂度的解决方法，兼顾自学教材和面试辅导的不同需求

数据结构与算法经典问题解析

（原书第2版）

[印度] 纳拉辛哈·卡鲁曼希（Narasimha Karumanchi）著
沈华 李兵兵 杜江毅 张明武 译

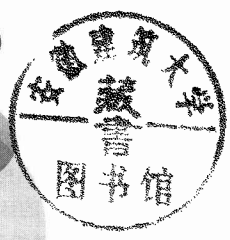
DATA STRUCTURES
AND ALGORITHMS MADE EASY
SECOND EDITION

机械工业出版社
China Machine Press

数据结构与算法经典问题解析

（原书第2版）

[印度] 纳拉辛哈·卡鲁曼希（Narasimha Karumanchi）著
沈华 李兵兵 杜江毅 张明武 译



DATA STRUCTURES
AND ALGORITHMS MADE EASY
SECOND EDITION

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

数据结构与算法经典问题解析（原书第2版）/（印）纳拉辛哈·卡鲁曼希（Narasimha Karumanchi）著；沈华等译．—北京：机械工业出版社，2018.10
书名原文：Data Structures and Algorithms Made Easy, Second Edition

ISBN 978-7-111-61241-4

I. 数… II. ①纳… ②沈… III. ①数据结构 ②算法分析 IV. ① TP311.12 ② TP312

中国版本图书馆CIP数据核字（2018）第241550号

本书版权登记号：图字 01-2016-0678

本书以问题驱动的方式系统地介绍了什么是数据结构，什么是算法，如何评价一个算法的好坏，以及各种常用的数据结构和常用的算法设计技巧。全书共21章，其覆盖的知识面很广泛，大致包括以下内容：基础知识（第1～2章，主要涉及数据结构的概念、算法的概念、算法分析的基本方法、递归等方面的内容）、常见的几种数据结构（第3～9章）、几种重要的算法（第10～14章）、常用的算法设计技巧（第15～19章）、基本的计算复杂性理论（第20章）和其他主题（第21章）。

本书可以作为高等院校计算机科学与技术及相关专业本科生的教材，也可以作为相关专业人员的参考资料。

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）
责任编辑：朱秀英　　责任校对：殷 虹
印　刷：北京诚信伟业印刷有限公司　　版　次：2019年1月第1版第1次印刷
开　本：186mm×240mm 1/16　　印　张：30.75
书　号：ISBN 978-7-111-61241-4　　定　价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066　　投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259　　读者信箱：hzjsj@hzbook.com

数据结构和算法是计算机科学的核心内容之一，是计算机专业学生的重要专业课程，对培养计算机专业设计与创新型人才起着关键作用。它具有概念多、内容广、抽象性强、需要一定的数学基础等特点，学习难度较大，对刚刚开始学习的学生来说更是如此。

在翻译 Narasimha Karumanchi 博士撰写的这本数据结构和算法入门书的过程中，我们发现该书有以下特点：

第一，语言通俗，与现实中的问题紧密相连，并且没有使用非常复杂的数学知识，非常适合数据结构和算法的初学者。

第二，对于经典的应用问题，常常给出基于不同数据结构的多种算法的求解方案，让读者充分了解和感受相同问题的不同算法在时间开销、空间开销和编程难度上的不同。

第三，习题丰富。本书的每一章都有包含大量习题的问题集，其中的每道题，作者均结合本章知识点或综合本章和其他章的知识点给出了分析和求解方案。

非常有幸得到机械工业出版社的信任参与了该书的翻译工作。由于水平有限，译文中表达不准确之处在所难免，还请业内专家和广大读者包涵，在此也敬请各位专家、学者和读者批评指正。欢迎各位读者将发现的错误或提出的任何意见和建议发送到邮箱 nancy78733@126.com。

最后，我要感谢湖北工业大学数据安全与隐私保护技术研究所、湖北工业大学工业大数据协同创新中心的诸位同事、同学的建议和帮助。在翻译过程中，张明武教授提出了很多有益的意见，在此表示诚挚的感谢。非常感谢机械工业出版社朱劼编辑对我的包容、理解、帮助和支持，也非常感谢机械工业出版社其他工作人员的辛勤付出。

沈　华

2018 年 3 月于武汉南湖

前　言

Data Structures and Algorithms Made Easy, Second Edition

我知道很多人不会阅读前言。但我强烈建议你至少浏览一遍本书的前言，因为你会发现一些与众不同的东西。

本书不是展示数据结构和算法的定理和证明，而是另辟蹊径，通过改进不同复杂度问题的解决方案来讲解数据结构和算法的相关内容(对于每一个问题，都有复杂度由大到小的多种解决方案)。这些方法基本上涵盖了所有可能的问题解决之道。这种讲解模式可以让你在遇到新问题时，获得思考所有可能解决方案的思维方式。本书也可以让你在准备面试、考试和升学面试时收益良多。

作为一名求职者，如果透彻理解了本书的内容，那么在面对面试官时将会从容不迫，甚至感到绰绰有余。作为一名教师，如果通读了本书，我相信你自然会提升教学质量，最终会让你的学生对选择了计算机科学/信息技术专业而感到庆幸。

对工程专业的本科生和研究生来说，本书在他们的学术准备中非常有用。本书每章都把重点放在问题和对问题的分析上，而不是单纯的理论。每章都是先介绍必需的基础理论，然后是问题集。本书大约有 700 个算法问题，并且给出了解决方法。

如果你是一名正在为参加计算机科学/信息技术竞赛考试做准备的学生，你会发现这本书涵盖了你所需要的所有主题，并且给出了详细的阐述。在写这本书的时候，我的关注点就已经放在了帮助参加这些考试的学生上。

对许多问题，本书给出了多种不同复杂度级别的解决方案。我们以蛮力法解决方案开始，慢慢地向可能的最优方案靠近。对每个问题，我们将设法弄清楚算法花费的时间和占用的内存空间。

本书建议的阅读方式是，先至少通读一遍以获得对所有主题的全面了解，随后读者可以按需跳到感兴趣的章节。尽管我们做了足够多的校验，书中难免还会存在一些不足。我们将在 www.careerMonk.com 网站上更新发现的任何错误。请读者关注该网站，后续任何错误修正、新问题和解决方案都会在该网站上更新。此外，请将你的宝贵意见发到下面的邮箱：Info@CareerMonk.com。

献上最美好的祝福。我相信你会发现此书物有所值。

Narasimha Karumanchi

绪　论

本章的目标是说明算法分析的重要性、算法分析需要用到的各种符号、这些符号之间的关系和运用算法分析的方法去解决尽可能多的问题。首先，让我们关注算法的基本要素、算法分析的重要性，然后，再逐渐介绍上述提到的其他主题。学完本章，对任何给定的算法(特别是递归函数)，读者应该能够给出它的复杂度。

1.1 变量

在给出变量的定义之前，让我们将它与古老的数学等式联系起来。从孩提时代开始，我们已经求解过许多数学等式。作为一个例子，考虑下面的等式：

$$x^2 + 2y - 2 = 1$$

我们不需要担心这个等式是否有使用价值。我们需要理解的关键问题是，这个等式中的一些名字(x 和 y)具有值(数据)。这意味着，这些名字(x 和 y)是用来表示数据的占位符。类似地，在计算机科学编程中，我们需要某些东西来代表数据，变量(variable)就扮演了这个角色。

1.2 数据类型

在上述数学等式中，变量 x 和 y 可以取任意值，如(10，20)范围内的整数、(0.23，5.5)范围内的实数，或者仅仅取值为 0 和 1。因此，为了求解这个等式，我们必须确定变量 x 和 y 取什么类型的值。在计算机科学编程中，数据类型(data type)就是用来实现这个目的的。在编程语言中，数据类型是预先定义了值的数据的集合，如整型、浮点型、字符型、字符串等。

计算机内存中存储的都是 0、1 代码串。如果我们想对某个问题进行编程求解，那么提供以 0、1 代码串表示的解决方案是非常困难的。为了解决这个困难，编程语言和编译器向我们提供了数据类型。例如，整型数据占 2 字节(这个值取决于具体的编译器)，浮点型数据占 4 字节，等等。这就是说，在内存中 2 字节(16 位)被合并了，称为一个整数。同样，4 字节(32 位)被合并了，称为一个浮点数。数据类型降低了编程的难度。在编程语言中，有两类数据类型：

- 系统定义的数据类型(也称为基本数据类型)
- 用户定义的数据类型(也称为用户自定义类型)

系统定义的数据类型(基本数据类型)

系统定义的数据类型被称为基本数据类型。许多编程语言提供以下基本数据类型：int、float、char、double、bool，等等。基本数据类型的数据在内存中占据的位数由编程语言、编译器和操作系统确定。即使是相同的基本数据类型，在不同的编程语言中它的

大小(即所占字节数)也可能不同。因为数据类型所有可能的取值(即取值范围)取决于它在内存中占据的位数，所以在不同编程语言中相同数据类型的取值范围也可能不同。

例如，int 类型数据的大小可能是 2 字节或者 4 字节。如果它的大小是 2 字节(16 位)，那么它的取值范围是(−32 768，+32 767)(即(-2^{15}，$2^{15}-1$))；如果它的大小是 4 字节(32 位)，那么它的取值范围是(−2 147 483 648，2 147 483 647)(即(-2^{31}，$2^{31}-1$))。其他数据类型也有相同的情况。

用户定义的数据类型(用户自定义类型)

如果系统定义的数据类型不能满足应用需求，那么一些编程语言允许用户定义自己的数据类型，这样的数据类型被称为*用户定义的数据类型*(或用户自定义类型)。C/C++ 语言中的结构(structure)和 Java 中的类(class)都是用户自定义类型的典型例子。例如，在下面的代码段中，我们运用结构体合并多个系统定义的数据类型生成了一个名为“newType”的用户自定义类型。这在处理计算机内存方面提供了更多的灵活性和舒适性。

```
struct newType {
    int data1;
    float data2;
    ...
    char data;
};
```

1.3　数据结构

基于上面的讨论可知，一旦有了变量中的数据，我们就需要一些使用这些数据解决问题的机制。在计算机中，*数据结构*(data structure)是以高效使用数据为目标的存储和组织数据的特定方式。一种*数据结构*就是组织和存储数据的一种特殊格式。常见的数据结构包括数组、文件、链表、栈、队列、树、图等。

从组织元素的角度，数据结构被分为以下两类：

1) *线性数据结构*：该数据结构中的元素按顺序依次进行访问，但并非一定按顺序依次进行存储。例如链表、栈和队列。

2) *非线性数据结构*：该数据结构中的元素以非线性的方式进行存储/访问。例如树和图。

1.4　抽象数据类型

给出抽象数据类型(ADT)的定义之前，让我们从另外一个角度去认识系统定义的数据类型。众所周知，默认情况下，所有基本数据类型(如 int、float 等)均支持诸如加法、减法这样的基本操作。系统提供这些基本数据类型的操作的实现。因此，对用户自定义数据类型而言，我们也需要定义它能够完成的操作。想使用用户自定义数据类型去解决实际应用问题，我们还需要给出这些操作的实现。这就意味着，通常用户自定义数据类型的定义包括对其操作的定义。

为了简化求解问题的过程，我们将数据结构和它的操作封装在一起称为*抽象数据类型*(Abstract Data Type，ADT)。一个抽象数据类型由两个部分构成：

1) 数据的声明

2) 操作的声明

常用的 ADT 包括：链表、栈、队列、优先队列、二叉树、字典、不相交集(合并和查找)、散列表、图等。例如，栈使用 LIFO(Last-In-First-Out)机制存储数据到栈中。最后入栈的元素最先出栈。栈的基本操作包括：栈的创建、入栈(即压入一个元素到栈顶)、出栈(即弹出栈顶元素)、访问当前的栈顶元素、求栈的长度(即栈中元素的个数)，等等。

在定义 ADT 时，我们无须关心其实现细节。只有当我们真正需要使用 ADT 时，才去考虑它们的实现细节。不同的 ADT 适用于不同的应用，适用于特定任务的 ADT 则是非常专业化的。通读完本书，读者会认识很多的 ADT，而且能够根据需要解决的应用问题选择合适的数据结构。

1.5　什么是算法

让我们考虑准备煎蛋的问题。为了煎蛋，我们会按以下步骤进行准备：

1) 准备煎锅；
2) 准备油；
    a. 有油吗？
        i. 如果有，那么将油倒入锅内；
        ii. 如果没有，那么我们需要去买油吗？
            1. 如果需要，那么我们去买油；
            2. 如果不需要，那么煎蛋终止；
3) 打开炉子；
……

对于一个给定问题(准备煎一个蛋)，我们所做的是，给出这个问题的一步一步的求解过程。算法的定义是：

> 算法是求解给定问题的一系列指令。

注意：我们不需要证明算法的每一个步骤。

1.6　为什么需要分析算法

为了实现从城市 A 移动到城市 B，我们有很多可选的方式：搭飞机、坐巴士、乘火车、骑自行车。根据可行性和便利性，我们会从中选择适合自己的一种。类似地，在计算机科学中，求解同一个问题有多种可选的算法(举个例子，排序问题有很多求解算法，如插入排序法、选择排序法、快速排序法，等等)。算法分析可以帮助我们判断哪个算法在时间和空间开销方面是高效的。

1.7　算法分析的目的

算法分析(analysis of algorithm)的目的是，主要从运行时间方面比较算法(或解决方案)，并且也从其他方面(如内存、开发人员的工作量等)对算法进行比较。

1.8　什么是运行时间分析

运行时间分析用来确定随着问题规模(即输入大小)的增大算法处理时间是如何增加的。输入大小是输入的元素个数。根据问题的类型，输入类型可以不同。下面是常见的

几种输入类型：

- 数组的大小
- 多项式的次数
- 矩阵的元素个数
- 输入的二进制表示的位数
- 图的顶点个数和边的条数

1.9　如何比较算法

为了比较算法，我们定义了如下几个客观度量(objective measure)：

执行时间。不是好的度量，因为执行时间依赖于具体的计算机。

执行的语句数。也不是好的度量，因为语句的个数依赖于具体的编程语言和程序员的个人风格。

理想的客观度量是什么？假设我们将给定算法的运行时间表示为关于输入大小 n 的函数(例如 $f(n)$)，那么比较算法就是比较与运行时间相关的不同函数。这种比较与具体的机器时间、编程风格等无关。

1.10　什么是增长率

算法的运行时间被表示为关于输入的函数，它随着输入规模增大的增长速度称为增长率(rate of growth)。假设你到一家车行去买一辆汽车和一辆自行车。如果你的朋友恰巧在车行碰到你并且询问你在买什么，那么通常你会回答在买汽车。为什么会这样？这是因为一辆汽车的花费远远大于一辆自行车的费用(即买自行车的费用被忽略在买汽车的费用中了)。

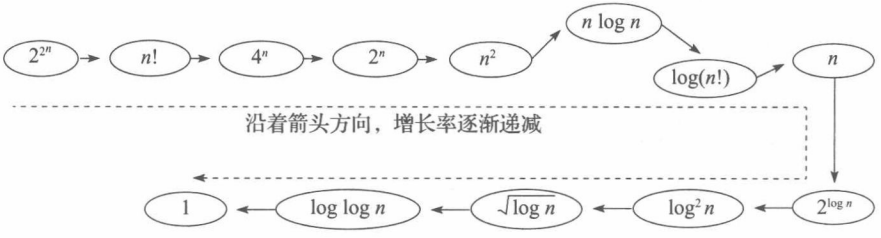
$$\text{Total Cost} = \text{cost\_of\_car} + \text{cost\_of\_cycle}$$
$$\text{Total Cost} \approx \text{cost\_of\_car(近似)}$$

我们可以用函数来刻画上述例子中买汽车和自行车的开销。对给定的函数可以忽略其低阶项，因为低阶项对整个函数的贡献相对来说微不足道(当输入规模 n 很大时)。在下面的例子中，n^4、$2n^2$、$100n$ 和 500 是某个函数的各项成本组成，可以将函数近似处理为 n^4，因为 n^4 是函数中增长速度最快的项。

$$n^4 + 2n^2 + 100n + 500 \approx n^4$$

1.11　常用的增长率

下图展示了不同增长率之间的关系。



下表列出了后续章节会涉及的增长率。

时间复杂度	名称	例子
1	常数阶	在链表首部插入一个元素
$\log n$	对数阶	在一个有序序列中查找元素
n	线性阶	在一个无序序列中查找元素
$n\log n$	线性对数阶	运用分治归并排序(即自顶向下的归并排序)对 n 个元素进行排序
n^2	平方阶	求图中两顶点之间的最短路径
n^3	立方阶	矩阵相乘
2^n	指数阶	汉诺塔问题

1.12　算法分析的类型

为了分析给定算法，我们需要知道算法在哪些输入下花费的时间较短(即表现良好)和在哪些输入下花费的时间较长。我们已经知道算法的时间开销可以表示成表达式的形式。这就意味着，我们可以用多个表达式来表达不同情况下算法的时间开销：需要较少时间的情况和需要较多时间的情况。

通常，第一种情况被称为算法的最好情况(best case)，第二种情况被称为算法的最坏情况(worst case)。为了分析算法，我们需要某种文法的基础和渐近分析/符号的基础。这里介绍三种类型的算法分析：

- **最坏情况**
  - 定义了算法需要最多时间的输入。
  - 该输入是使算法运行最慢的输入。
- **最好情况**
  - 定义了算法需要最少时间的输入。
  - 该输入是使算法运行最快的输入。
- **平均情况**
  - 提供了关于算法运行时间的预测。
  - 假设输入是随机的。

$$\text{下界} \leqslant \text{平均时间} \leqslant \text{上界}$$

对一个给定的算法，我们可以用表达式来分别表示它在最好情况、最坏情况和平均情况下的时间开销。例如，令 $f(n)$ 是表示某算法时间开销的函数。

$$f(n) = n^2 + 500, \text{最坏情况}$$
$$f(n) = n + 100n + 500, \text{最好情况}$$

平均情况也是类似的，表达式定义了算法需要花费平均运行时间(或内存空间)的那些输入。

1.13　渐近符号

现在我们已经得到了在最好情况、平均情况和最坏情况下算法时间开销(或空间开销)的表达式。下面我们需要做的是，确定这三种情况下算法开销的上界和下界。为此，

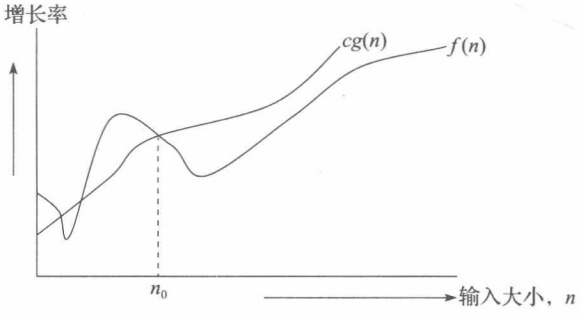
我们需要用某种文法来描述这些界限。为了方便后面几个小节对这个主题的讨论，我们假设某算法的(时间或空间)开销为 $f(n)$。

1.14　O 符号

O 符号给出了函数的紧致上界。它通常被描述为 $f(n)=O(g(n))$，表示对于一个很大的 n，$f(n)$的上界是 $g(n)$。例如，如果 $f(n)=n^4+100n^2+10n+50$，那么 $g(n)=n^4$。这意味着，对于一个很大的 n，$g(n)$给出了 $f(n)$的最大增长率。

下面让我们更加详细地了解 O 符号。O 符号定义为：$O(g(n))=\{f(n)$：存在一个正常数 c 和正整数 n_0，使得对所有 $n\geqslant n_0$ 有 $0\leqslant f(n)\leqslant cg(n)$成立$\}$。$g(n)$是 $f(n)$的渐近紧致上界。我们的目标是给出不小于算法增长率 $f(n)$的最小增长率 $g(n)$。

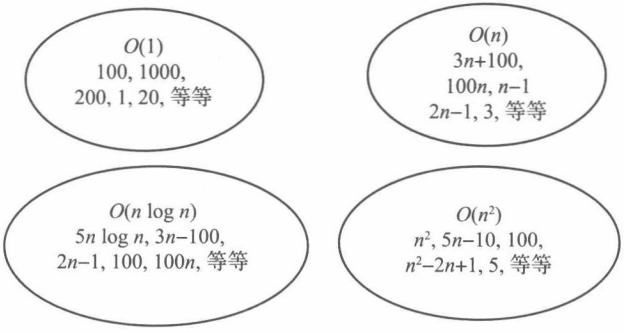
因为 n 较小时的增长率是没有太大意义的，所以通常我们会忽略掉它。在下图中，n_0 是我们考虑某算法增长率的分界点。可见，n 较小时(即小于 n_0)的增长率可能与 n 较大时(即大于 n_0)的增长率不同。



O 符号的图形化演示

$O(g(n))$是一个函数的集合，这个集合中的任意一个函数具有比 $g(n)$小或相同阶的增长率。例如，$O(n^2)$包括 $O(1)$、$O(n)$、$O(n\log n)$，等等。

注意：我们仅仅进行 n 取大值时的算法分析。也就是说，我们不考虑小于 n_0 的算法增长率。



关于 O 符号的例子

例 1　求 $f(n)=3n+8$ 的上界。

求解：对所有 $n \geqslant 8$ 有 $3n+8 \leqslant 4n$，所以 $3n+8=O(n)$，其中 $c=4$，$n_0=8$。

例 2　求 $f(n)=n^2+1$ 的上界。

求解：对所有 $n \geqslant 1$ 有 $n^2+1 \leqslant 2n^2$，所以 $n^2+1=O(n^2)$，其中 $c=2$，$n_0=1$。

例 3　求 $f(n)=n^4+100n^2+50$ 的上界。

求解：对所有 $n \geqslant 11$ 有 $n^4+100n^2+50 \leqslant 2n^4$，所以 $n^4+100n^2+50=O(n^4)$，其中 $c=2$，$n_0=11$。

例 4　求 $f(n)=2n^3-2n^2$ 的上界。

求解：对所有 $n \geqslant 1$ 有 $2n^3-2n^2 \leqslant 2n^3$，所以 $2n^3-2n^2=O(2n^3)$，其中 $c=2$，$n_0=1$。

例 5　求 $f(n)=n$ 的上界。

求解：对所有 $n \geqslant 1$ 有 $n \leqslant n$，所以 $n=O(n)$，其中 $c=1$，$n_0=1$。

例 6　求 $f(n)=410$ 的上界。

求解：对所有 $n \geqslant 1$ 有 $410 \leqslant 410$，所以 $410=O(1)$，其中 $c=1$，$n_0=1$。

不是唯一的解？

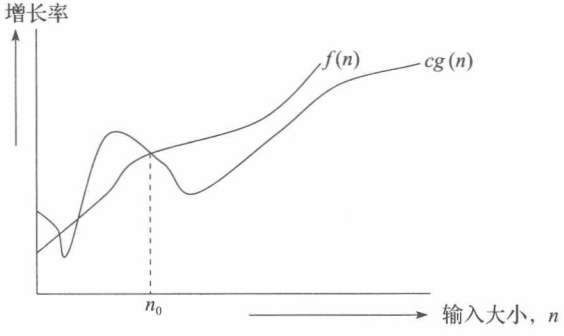
在给出渐近界限的时候，n_0 和 c 的取值并不唯一。例如，$100n+5=O(n)$。求解过程中 n_0 和 c 有多个可能的取值。

求解 1：对所有 $n \geqslant 5$ 有 $100n+5 \leqslant 100n+n=101n \leqslant 101n$，因此 $c=101$，$n_0=5$ 是一个可能的解。

求解 2：对所有 $n \geqslant 1$ 有 $100n+5 \leqslant 100n+5n=105n \leqslant 105n$，因此 $c=105$，$n_0=1$ 也是一个可能的解。

1.15　Ω 符号

类似于 O 符号的讨论，Ω 符号给出了函数的紧致下界。它通常被描述为 $f(n)=\Omega(g(n))$，表示对于一个很大的 n，$f(n)$的紧致下界是 $g(n)$。例如，如果 $f(n)=100n^2+10n+50$，那么 $g(n)=\Omega(n^2)$。



Ω 符号可以定义为：$\Omega(g(n))=\{f(n)$：存在一个正常数 c 和正整数 n_0，使得对所有 $n \geqslant n_0$ 有 $0 \leqslant cg(n) \leqslant f(n)$ 成立$\}$。$g(n)$是 $f(n)$的渐近紧致下界。我们的目标是给出不大于算法增长率 $f(n)$的最大增长率 $g(n)$。

关于 Ω 符号的例子

例 1　求 $f(n)=5n^2$ 的下界。

求解： $\exists c$ 和 n_0 使得 $0 \leqslant cn \leqslant 5n^2 \Rightarrow cn \leqslant 5n^2 \Rightarrow c=1$，$n_0=1$。

所以 $5n^2=\Omega(n^2)$，其中 $c=1$，$n_0=1$。

例 2　证明 $f(n)=100n+5 \neq \Omega(n^2)$。

求解： $\exists c$ 和 n_0 使得 $0 \leqslant cn^2 \leqslant 100n+5$，又 $100n+5 \leqslant 100n+5n(\forall n \geqslant 1)=105n$，故 $cn^2 \leqslant 105n \Rightarrow n(cn-105) \leqslant 0$；

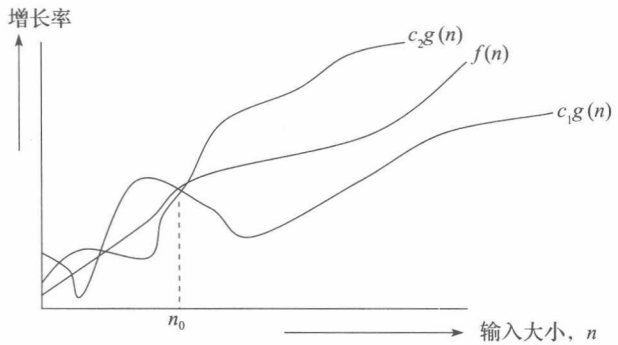
因为 n 是正整数$\Rightarrow cn-105 \leqslant 0 \Rightarrow n \leqslant 105/c \Rightarrow$ 矛盾：n 不可能比某一个常数小。

例 3　$2n=\Omega(n)$，$n^3=\Omega(n^3)$，$\log n=\Omega(\log n)$。

1.16　Θ 符号

Θ 符号可以确定函数(或算法)的上界和下界是否相同。算法的平均运行时间总是介于下界和上界之间的。如果上界(O)和下界(Ω)给出相同的结果，那么 Θ 符号也会给出同样的结果。举个例子，假设有 $f(n)=10n+n$。那么，$f(n)$的紧致上界 $g(n)$是 $O(n)$，并且最好情况下的增长率 $g(n)=O(n)$。

在这个例子中，最好情况和最坏情况下的增长率相同。因此，平均情况下也应该相同。对一个给定的函数(或算法)，如果 O 和 Ω 表示的增长率(即界限)不相同，那么 Θ 表示的增长率就可能与它们不同。在这种情况下，我们需要考虑针对函数(或算法)所有可能输入的时间复杂度，然后求出它们的均值(例如，快速排序的平均情况，请参见第10章)。



现在给出 Θ 符号的定义：$\Theta(g(n))=\{f(n)$：存在两个正常数 c_1、c_2 和正整数 n_0 使得对所有 $n \geqslant n_0$ 有 $0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$成立$\}$。$g(n)$是 $f(n)$的渐近紧致界。$\Theta(g(n))$是所有增长率与 $g(n)$同阶的函数构成的集合。

关于 Θ 符号的例子

例 1　求 $f(n)=\frac{n^2}{2}-\frac{n}{2}$的 Θ 界。

求解： 对所有 $n \geqslant 1$ 有 $\frac{n^2}{5} \leqslant \frac{n^2}{2}-\frac{n}{2} \leqslant n^2$，所以 $\frac{n^2}{2}-\frac{n}{2}=\Theta(n^2)$，其中 $c_1=1/5$，$c_2=1$，$n_0=1$。

例 2　证明 $n \neq \Theta(n^2)$。

求解： $c_1 n^2 \leqslant n \leqslant c_2 n^2 \Rightarrow$仅当 $n \leqslant 1/c_1$ 时不等式成立，所以 $n \neq \Theta(n^2)$。