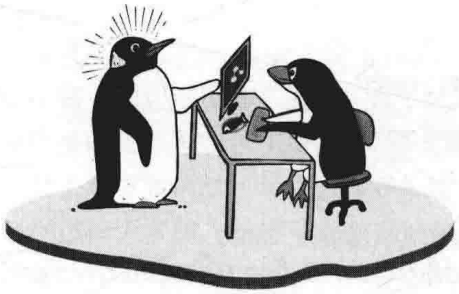
简单有趣的算法入门书

图解算法

使用Python

吴灿铭 胡昭民 / 著

清華大學出版社



图解算法

使用Python

吴灿铭 胡昭民 / 著

清華大學出版社
北京

内 容 简 介

本书是一本综合讲述数据结构及其算法的入门书，力求简洁、清晰、严谨且易于学习和掌握，并没有追求大而全的数据结构和所有相关的算法，而是选择经典的算法来配合介绍常用的数据结构，包括数组、链表、堆栈、队列以及树和图等。

本书为每个算法及其数据结构均提供了演算的详细图解，并为每个经典的算法都提供了 Python 语言编写的完整范例程序（包含完整的源代码）。每个范例程序都经过了测试和调试，可以直接在标准的 Python 解释器中运行，非常适合作为普及型的教科书或自学读物。

图书在版编目（CIP）数据

图解算法：使用 Python / 吴灿铭，胡昭民著. —北京：清华大学出版社，2018
ISBN 978-7-302-50988-2

Ⅰ.①图… Ⅱ.①吴… ②胡… Ⅲ.①计算机算法－图解 Ⅳ.①TP301.6-64

中国版本图书馆 CIP 数据核字（2018）第 192202 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：北京国马印刷厂
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**11.25　　**字　　数：**288 千字
版　　次：2018 年 10 月第 1 版　　**印　　次：**2018 年 10 月第 1 次印刷
定　　价：49.00 元

产品编号：080316-01

序

本书是一本综合讲述数据结构及其算法的入门书，力求简洁、清晰、严谨且易于学习和掌握。本书的目标不是追求大而全的数据结构和所有相关的算法，而是选择经典的算法来配合介绍常用的数据结构。

本书针对的读者主要是中学、专科学校的学生，或者是非信息类专业的学生。因为信息专业类学习用的“数据结构与算法”内容更深，所以本书不适合作为这类专业的教科书使用，而更加适合普及型的教科书或自学读物。

为了便于学校的教学或者读者自学，作者在描述数据结构原理和算法时为每个算法及数据结构提供了演算的详细图解。另外，为了适合教学中让学生上机实践或者自学者上机“操练”，本书为每个经典的算法都提供了 Python 语言编写的完整范例程序（包含完整的源代码），每个范例程序都经过了测试和调试，可以直接在标准的 Python 解释器中运行，目的就是让本书的学习者以这些范例程序作为参照，迅速掌握数据结构和算法的要点。本书所有范例程序下载的网址如下：

https://pan.baidu.com/s/1lvf3InAb2XEpwMDhOsNIhw（注意区分数字与字母大小写）

还可以扫描下面的二维码进行下载。若下载有问题，请电子邮件联系 booksaga@126.com，邮件标题为“求代码，图解算法——使用 Python”。

学习本书需要有面向对象程序设计语言的基础，如果读者没有学习过任何面向对象的程序设计语言，那么建议读者先学习一下 Python 语言再来学习本书。如果读者已经掌握了 Java、C++、C#等任何一种面向对象的程序设计语言，即便没有学习过 Python 语言，则只需找一本“Python 语言快速入门”方面的参考书快速浏览一下，即可开始本书的学习。

如果是用于自学的，则可以从 https://www.python.org/ 网站中下载适合于本地计算机及其操作系统的 Python 版本，再安装和设置好 Python 标准运行环境。

资深架构师　赵军

2018 年 8 月

前　言

如今“程序设计”已经是越来越普及的课程，让人人拥有程序设计的能力已是各所学校信息课程的重点。算法一直是计算机科学专业非常重要的基础课程，从程序设计语言实践的角度而言，算法是有志从事信息技术领域工作的专业人员必须学习的一门基础理论课程。无论你采用哪一种程序设计语言编写程序，所设计的程序能否快速而高效地完成预定的任务，算法都是其中的关键因素。

市面上有许多算法相关的书籍，常会介绍大量的理论或是在书上举例去表达算法的核心概念，虽然有些书写作的笔法轻松，能帮助用户理解各种算法的核心概念，但是这类书缺乏完整的程序设计语言的实现范例，因而对于第一次接触算法的初学者来说，将算法运用于实际应用就成了一大段跨不过去的鸿沟。

为了帮助更多人用比较轻松的方式了解各种算法的重点，包括分治法、递归法、贪心法、动态规划法、迭代法、枚举法、回溯法等，以及应用不同算法所延伸出的重要数据结构（例如数组、链表、堆栈、队列、树形结构、图形、排序、查找、哈希等），本书特别采用丰富的图例来阐述算法的基本概念，并将算法概念进行言简意赅的诠释和举例，同时使用 Python 语言编程实现算法，以期能将各种算法真正应用在学习者将来的程序设计中。因此，这是一本学习算法的入门教科书。

笔者长期从事信息教育以及专业书籍编写的工作，因而在语句的表达上尽量简洁有力。另外，为了检验各章的学习成果，特别收集了难易度适中的习题，并参阅算法与数据结构课程考试的相关题型，让读者进一步演练与巩固算法的基础知识。然而，一本好的算法教科书，除了内容的完备专业外，更需要有清楚易懂的结构安排及表达方式。希望本书可以帮助读者在轻松的学习氛围下，对算法这门基础理论有比较深刻的认识。

吴灿铭

目　录

第1章

进入算法的世界

计算机（computer），或者被人们称为电脑，是一种具备了数据计算与信息处理功能的电子设备。对于一个有志于从事信息技术专业领域的人员来说，程序设计是一门和计算机硬件与软件息息相关的课程，称得上是从计算机问世以来经久不衰的热门学科。

随着信息与网络科技的高速发展，在目前这个物联网（Internet of Things，IOT）与云运算（Cloud Computing）的时代（见图 1-1），程序设计能力已经被看成是国力的象征，有条件的中小学校都将程序设计（或称为“编程”）列入学生信息课的学习内容，在大专院校里程序设计则不再只是信息技术相关院系的“专利”了。程序设计已经是接受全民义务教育的学生们所应该具备的基本能力，只有将“创意”经由“设计过程”与计算机相结合，才能让新一代人才轻松应对这个快速变迁的云计算时代。

图 1-1　云计算加速了全民进入程序设计的时代

没有最好的程序设计语言，只有是否适合的程序设计语言。程序设计语言本来就只是工具，

从来都不是算法的重点，我们知道一个程序能否快速而高效地完成预定的任务，算法才是其中关键的因素。本章将介绍算法的基本概念和算法性能的分析，并介绍一些基本的数据结构，以作为往后章节讨论的基础，让读者逐步认识算法。

提示

“云”其实就是泛指“网络”，因为工程师在网络结构示意图中通常习惯用“云朵状”图来代表不同的网络。云运算是指将网络中运算能力提供出来作为一种服务，只要用户可以通过网络登录远程服务器进行操作，就能使用这种运算资源。

物联网（Internet of Things，IOT）是近年来信息产业中一个非常热门的话题，各种配备了传感器的物品，例如 RFID、环境传感器、全球定位系统（GPS）等，与因特网结合起来，并通过网络技术让各种实体对象、自动化设备彼此沟通和交换信息，也就是通过网络把所有东西都连接在一起。

1.1 生活中到处都是算法

算法（algorithm）是计算机科学中程序设计领域的核心理论之一。每个人每天都会用到一些算法，算法也是人类使用计算机解决问题的技巧之一，但是算法并不是仅仅用于计算机领域中，包括在数学、物理甚至是每天的生活中都应用广泛。在日常生活中就有许多工作都可以使用算法来描述，例如员工的工作报告、宠物的饲养过程、厨师准备美食的食谱、学生的课程表等，如今我们几乎每天都要使用的各种搜索引擎都必须借助不断更新的算法来运行（见图 1-2）。

图 1-2 搜索引擎的运行也必须借助不断更新的算法

特别是在算法与大数据的结合下，这门学科演化出“千奇百怪”的应用，例如当我们拨打某个银行信用卡客户服务中心的电话时，很可能就先经过后台算法的过滤，帮我们找出一名最“合我

们胃口”的客服人员来与我们交谈。在互联网时代，通过大数据分析，网店还可以进一步了解产品购买和需求的人群是哪些一类人，甚至一些知名 IT 企业在面试过程中也会测验新进人员对于算法的了解程度（见图 1-3）。

图 1-3　一些知名 IT 企业面试也会测验对算法的了解程度

提示

大数据（Big Data，又称海量数据），由 IBM 于 2010 年提出，是指在一定时效（Velocity）内进行大量（Volume）、多样性（Variety）、低价值密度（Value）、真实性（Veracity）数据的获得、分析、处理、保存等操作，主要特性包含 5 个方面：Volume（大量）、Velocity（时效性）、Variety（多样性）、Value（低价值密度）、Veracity（真实性）。由于数据的来源有非常多的途径，大数据的格式也越来越复杂，大数据解决了商业智能无法处理的非结构化与半结构化数据。

1.1.1　算法的定义

在韦氏辞典中算法定义为：“在有限步骤内解决数学问题的程序。”如果运用在计算机领域中，我们也可以把算法定义成：“为了解决某项工作或某个问题，所需要有限数量的机械性或重复性指令与计算步骤。”

我们知道可整除两个整数的最大整数被称为这两个整数的最大公约数，而辗转相除法可以用来求取两个整数的最大公约数，即可以使用这个辗转相除法的算法来求解。下面我们使用 while 循环来设计一个 Python 程序，求取所输入两个整数的最大公约数（g.c.d）。辗转相除法的 Python 算法如下：

```
Num_1=int(input('请输入第一个整数: '))
Num_2=int(input('请输入第二个整数: '))

if Num_1 < Num_2:
    Tmp_Num=Num_1
    Num_1=Num_2
    Num_2=Tmp_Num

while Num_2 != 0:
```

```
        Tmp_Num=Num_1 % Num_2
        Num_1=Num_2
        Num_2=Tmp_Num

    print('最大公约数(g.c.d): ',Num_1)
```

1.1.2 算法的条件

在计算机系统中算法更是不可或缺的一环，在这里要讨论计算机程序常使用到的算法的概念与定义。认识了算法的定义之后，我们再来说明一下算法所必须符合的五个条件，如图 1-4 和表 1-1 所示。

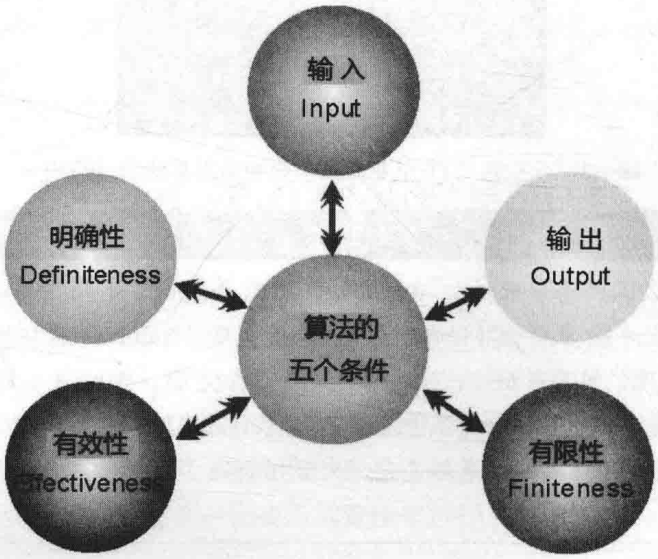


图 1-4 算法必须符合的五个条件

表 1-1 算法必须符合的五个条件

算法的特性	内容与说明
输入（Input）	0 个或多个输入数据，这些输入必须有清楚的描述或定义
输出（Output）	至少会有一个输出结果，不可以没有输出结果
明确性（Definiteness）	每一个指令或步骤必须是简洁明确的
有限性（Finiteness）	在有限步骤后一定会结束，不会产生无限循环
有效性（Effectiveness）	步骤清楚且可行，能让用户用纸笔计算而求出答案

我们认识了算法的定义与条件后，接着要来思考：该用什么方法来表达算法最为适当呢？其实算法的主要目的在于让人们了解所执行的工作流程与步骤，只要能清楚地体现算法的五个条件即可。常用的算法如下：

常用的算法一般可以用中文、英文、数字等文字来描述，即用语言来描述算法的具体步骤。例如，小华早上去上学并买早餐的简单文字算法如图 1-5 所示。

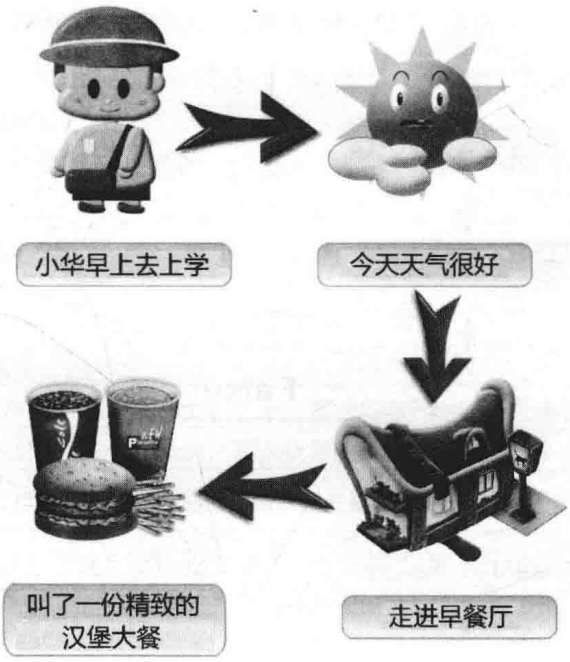


图 1-5　文字描述的算法例子

算法可以用可读性高的高级语言或伪语言（Pseudo-Language）来描述或者表达。以下算法是用 Python 语言描述的，它是计算所传入的两数 x、y 的 x^y 值函数 Pow()：

```
def Pow(x,y):
    p=1
    for i in range(1,y+1):
        p *=x
    return p

print(Pow(4,3))
```

提示

伪语言（Pseudo-Language）接近高级程序设计语言，也是一种不能直接放进计算机中执行的语言。一般都需要一种特定的预处理器（preprocessor），或者要用人工编写转换成真正的计算机语言，经常使用的有 SPARKS、PASCAL-LIKE 等语言。

流程图（Flow Diagram）是一种通用的以图形符号来表示程序执行流程的工具，也是常用描述算法的工具。例如，请用户输入一个数值，然后判断这个数值是奇数还是偶数，这个算法描述的流程图如图 1-6 所示。

提示

算法和过程（procedure）有何不同？因为过程不一定要满足有限性的要求，例如操作系统或机器上运作的过程。除非宕机，否则永远在等待循环中（waiting loop），这就违反了算法五大条件中的“有限性”。

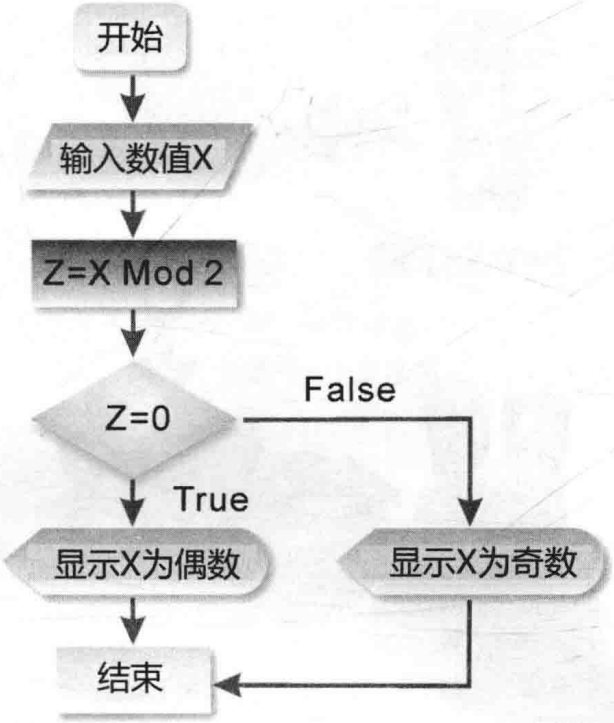


图 1-6 用流程图描述算法的例子

1.1.3 时间复杂度 O(f(n))

大家可能会想，应该怎么评估一个算法的好坏呢？例如，程序设计人员可以把某个算法的执行步骤计数来作为衡量运行时间的标准，例如同样是两条程序语句：

```
a=a+1 与 a=a+0.3/0.7*10005
```

由于涉及变量存储类型与表达式的复杂度，因此真正绝对精确的运行时间一定不相同。不过话又说回来，如此大费周章地去精确计算程序的运行时间往往寸步难行，而且也毫无意义。因此用一种“估算”的方法来衡量程序或算法的运行时间反而更加恰当，这种估算的时间就是“时间复杂度”（Time Complexity)。详细定义如下：

在一个完全理想状态下的计算机中，我们定义 T(n)来表示程序执行所要花费的时间，其中 n 代表数据输入量。当然程序的运行时间（Worse Case Executing Time）或最大运行时间是时间复杂度的衡量标准，一般以 Big-oh 表示。

在分析算法的时间复杂度时，往往用函数来表示它的成长率（Rate of Growth），其实时间复杂度是一种“渐近表示法”（Asymptotic Notation）。

O(f(n))可视为某算法在计算机中所需运行时间不会超过某一常数倍数的 f(n)，也就是说某算法运行时间 T(n)的时间复杂度（time complexity）为 O(f(n))（读成 Big-oh of f(n)或 Order is f(n)）。

意思是存在两个常数 c 与 n_0，当 $n \geq n_0$ 时，$T(n) \leq cf(n)$。f(n)又被称为运行时间的成长率（rate of growth）。由于采用的是宁可高估也不要低估的原则，因此估计出来的函数是真正所需运行时间的上限。请大家看以下的范例，以便更好地了解时间复杂度的含义。

范例：运行时间 $T(n)=3n^3+2n^2+5n$，时间复杂度是多少？

答：首先得找出常数 c 与 n_0。我们可以找到当 $n_0=0$、$c=10$ 时，若 $n \geq n_0$，则 $3n^3+2n^2+5n \leq 10n^3$，因此得知时间复杂度为 $O(n^3)$。

事实上，时间复杂度只是执行次数的一个概略的估算，并非真实的执行次数。而 Big-oh 是一种用来表示最坏运行时间的估算方式，也是最常用于描述时间复杂度的渐近式表示法。常见的 Big-oh 如表 1-2 和图 1-7 所示。

表 1-2　常见的 Big-oh

Big-oh	特色与说明
O(1)	称为常数时间（constant time），表示算法的运行时间是一个常数倍数
O(n)	称为线性时间（linear time），表示执行的时间会随着数据集合的大小而线性增长
$O(\log_2 n)$	称为次线性时间（sub-linear time），成长速度比线性时间还慢，而比常数时间快
$O(n^2)$	称为平方时间（quadratic time），算法的运行时间会成二次方的增长
$O(n^3)$	称为立方时间（cubic time），算法的运行时间会成三次方的增长
(2^n)	称为指数时间（exponential time），算法的运行时间会成 2 的 n 次方增长。例如，解决 Nonpolynomial Problem 问题算法的时间复杂度即为 $O(2^n)$
$O(n\log_2 n)$	称为线性乘对数时间，介于线性和二次方增长的中间模式

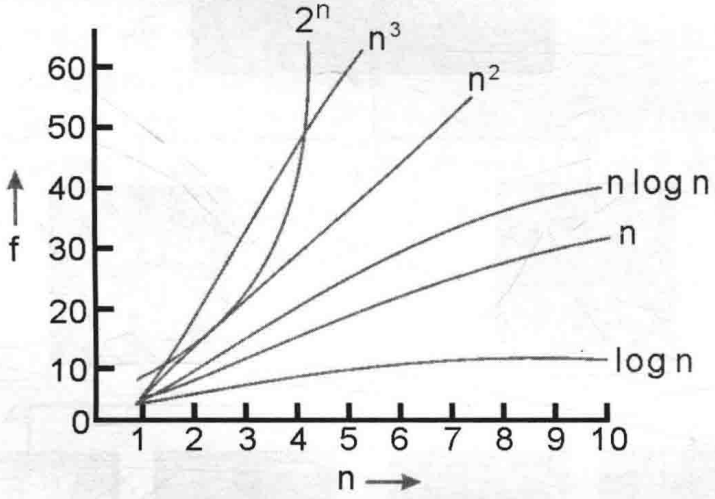


图 1-7　常见的最坏运行时间曲线

对于 $n \geq 16$ 时，时间复杂度的优劣比较关系如下：

$$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)$$

1.2　常见算法简介

善用算法，当然是培养程序设计逻辑很重要的步骤，许多实际的问题都可用多个可行的算法来解决，但是要从中找出最佳的解决算法却是一项挑战。本节中将为大家介绍一些近年来相当知名

的算法，能帮助大家更加了解不同算法的概念与技巧，以便日后更有能力分析各种算法的优劣。

1.2.1 分治法

分治法（Divide and Conquer，也称为“分而治之法”）是一种很重要的算法，我们可以应用分治法来逐一拆解复杂的问题，核心思想就是将一个难以直接解决的大问题依照相同的概念，分割成两个或更多的子问题，以便各个击破，即“分而治之”。其实，任何一个可以用程序求解的问题所需的计算时间都与其规模有关，问题的规模越小，越容易直接求解。分割问题也是遇到大问题的解决方式，可以使子问题规模不断缩小，直到这些子问题足够简单到可以解决，最后再将各子问题的解合并得到原问题的最终解答。这个算法应用相当广泛，如快速排序法（quick sort）、递归算法（recursion）、大整数乘法。

下面我们就以一个实际的例子来说明。如果有 8 幅很难画的图，我们可以分成 2 组各四幅画来完成，如果还是觉得太复杂，继续再分成四组，每组各两幅画来完成，采用相同模式反复分割问题，这就是最简单的分治法的核心思想，如图 1-8 所示。

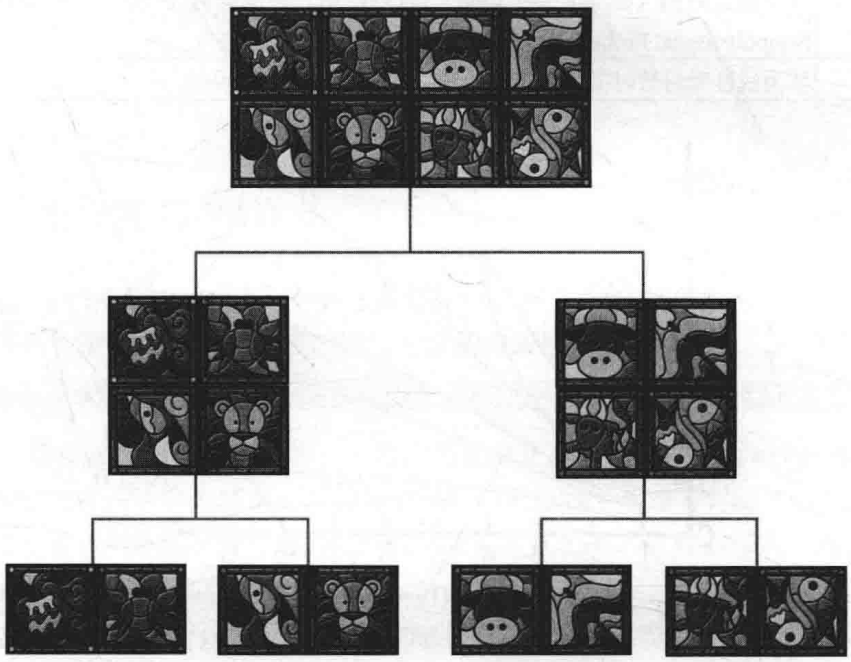


图 1-8 分治法算法的例子之一

再举个例子，如果你被委派做一个项目的规划，这个项目规划有 8 个章节的主题，如果只靠一个人独立完成，不仅时间比较长，而且有些规划的内容也有可能不是自己的专长，这个时候就可以按照这 8 个章节的特性分工给 2 位项目负责人去完成。不过，为了让这个规划更快完成，又能找到适合的分类，再分别将其分割成 2 章，并分派给更多不同的项目成员，如此一来，每位成员只需负责其中 2 个章节，经过这样的分配，就可以将原先的大项目简化成 4 个小项目，并委派给 4 个成员去完成。以此类推，根据分治法的核心思想，又可以将其切割成 8 个小主题，再委派给 8 个成员去分别完成，因为参与人员较多，所以所需时间缩减到原先一个人独立完成的时间。这个例子的分治法解决方案的示意图如图 1-9 所示。

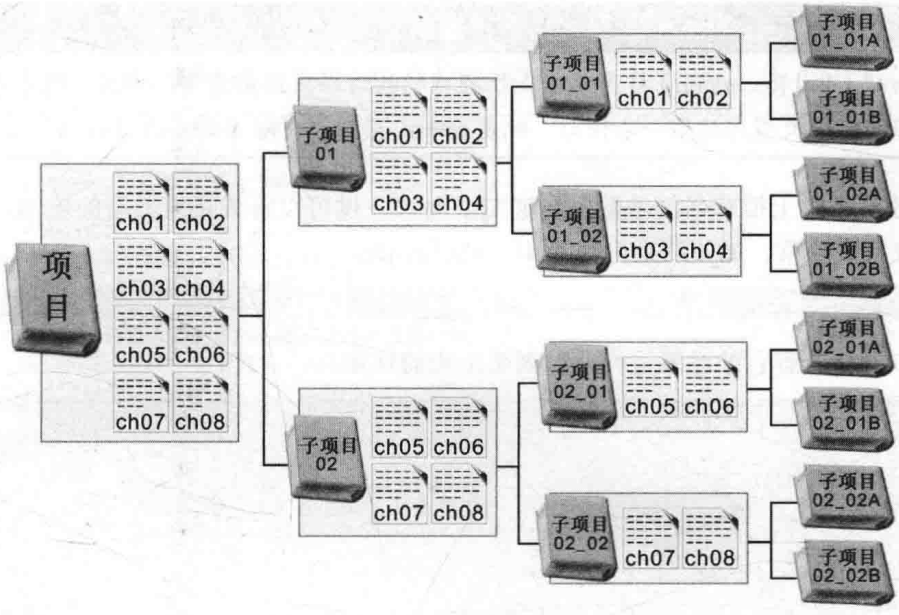


图 1-9　分治法算法的例子之二

分治法也可以应用在数字的分类与排序上，如果以人工的方式将散落在地上的打印稿，按从第 1 页整理并排序到第 100 页，我们可以采用两种方法。

（1）逐一捡起打印稿，并逐一按页码顺序插入到正确的位置。但是这样的方法有一种缺点，就是排序和整理的过程较为繁杂，而且较为花时间。

（2）应用分治法的原理，先行将页码 1 到页码 10 放在一起，页码 11 到页码 20 放在一起，以此类推，最后将页码 91 到页码 100 放在一起，也就是说，将原先的 100 页分类为 10 个页码区间，然后我们分别对 10 堆页码进行整理，最后按页码从小到大的分组合并起来，轻易恢复到原先的稿件顺序，通过分治法可以让原先复杂的问题变成规则更简单、数量更少、速度更快且更容易轻易解决的小问题。

1.2.2　递归法

递归是一种很特殊的算法，分治法和递归法很像一对孪生兄弟，都是将一个复杂的算法问题进行分解，让规模越来越小，最终使子问题容易求解。递归在早期人工智能所用的语言中，如 Lisp、Prolog 几乎是整个语言运行的核心，现在许多程序设计语言，包括 C、C++、Java、Python 等，都具备递归功能。简单来说，对程序设计人员的实现而言，“函数”（或称为子程序）不单纯只是能够被其他函数调用（或引用）的程序单元，在某些程序设计语言中还提供了自己调用自己的功能，这种调用功能就是所谓的“递归”。

从程序设计语言的角度来说，递归的正式定义为：一个函数或子程序，是由自身所定义或调用的，就称为递归（Recursion）。递归至少要满足 2 个条件：一个可以反复执行的递归过程；一个可以离开递归执行过程的出口。