HZ BOOKS
华章教育

计 算 机 科 学 丛 书

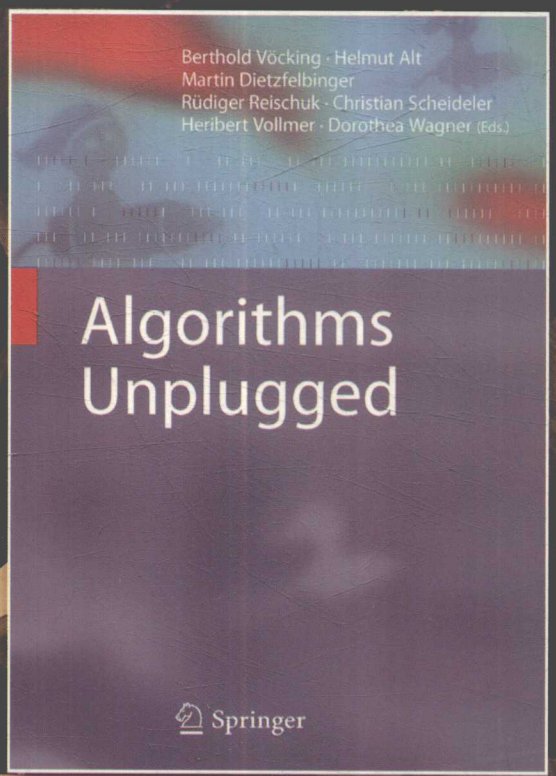
Springer

无处不在的算法

贝特霍尔德·弗金（Berthold Vöcking） 赫尔穆特·阿尔特（Helmut Alt）
马丁·迪茨费尔宾格（Martin Dietzfelbinger） 吕迪格·赖舒科（Rüdiger Reischuk）
[德] 克里斯蒂安·沙伊德勒（Christian Scheideler） 黑里贝特·沃尔默（Heribert Vollmer） 编著
多萝西娅·瓦格纳（Dorothea Wagner）

陈道蓄
南京大学 译

Algorithms Unplugged



机械工业出版社
China Machine Press

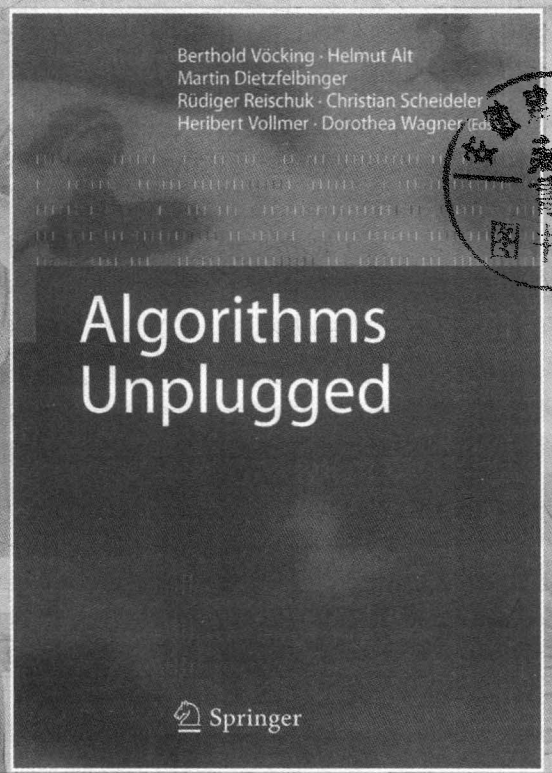
计 算 机 科 学 丛 书

无处不在的算法

贝特霍尔德·弗金（Berthold Vöcking） 赫尔穆特·阿尔特（Helmut Alt）
马丁·迪茨费尔宾格（Martin Dietzfelbinger） 吕迪格·赖舒科（Rüdiger Reischuk）
[德] 克里斯蒂安·沙伊德勒（Christian Scheideler） 黑里贝特·沃尔默（Heribert Vollmer） 编著
多萝西娅·瓦格纳（Dorothea Wagner）

陈道蓄 译
南京大学

Algorithms Unplugged



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

无处不在的算法 /（德）贝特霍尔德·弗金（Berthold Vòcking）等编著；陈道蓄译 . —北京：机械工业出版社，2018.9
（计算机科学丛书）
书名原文：Algorithms Unplugged

ISBN 978-7-111-60869-1

I. 无… II. ① 贝… ② 陈… III. 算法 - 高等学校 - 教材 IV. O24

中国版本图书馆 CIP 数据核字（2018）第 216581 号

本书版权登记号：图字 01-2012-8919

本书以简单易懂的写作风格，通过解决现实世界常见的问题来介绍各种算法技术，揭示了算法的设计与分析思想。全书共有 41 章，分为四大部分，图文并茂，把各种算法的核心思想讲得浅显易懂。

本书既可作为面向青少年的科普读物，也可作为高等院校算法相关课程的本科生教材，还可供教师和专业技术人员参考阅读。

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：唐晓琳　　责任校对：殷 虹
印　刷：北京市兆成印刷有限责任公司　　版　次：2018 年 10 月第 1 版第 1 次印刷
开　本：185mm × 260mm 1/16　　印　张：17
书　号：ISBN 978-7-111-60869-1　　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

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson、McGraw-Hill、Elsevier、MIT、John Wiley & Sons、Cengage 等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出 Andrew S. Tanenbaum、Bjarne Stroustrup、Brian W. Kernighan、Dennis Ritchie、Jim Gray、Afred V. Aho、John E. Hopcroft、Jeffrey D. Ullman、Abraham Silberschatz、William Stallings、Donald E. Knuth、John L. Hennessy、Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近伍百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章科技图书出版中心

译 者 序

Algorithms Unplugged

我问一些刚刚进入大学的新生为什么选择计算机类专业，几乎每个人都提到对“人工智能”感兴趣，但似乎其中多数人对人工智能并没有多少了解。

本书第41章讨论“模拟退火”算法，用来作为例子的是一个几何拼图类的组合游戏，大多数人会将其视为一种“智力游戏”。当实验证明计算机可以得到比绝大多数人更高的分数，这是否就体现了计算机的“智能”呢？

一个有好奇心的中学生不难理解第41章的内容。其实机器并不“知道”自己在做什么，它只是执行几个一点都不复杂的动作。告诉机器以什么样的顺序去执行某些（其实很基本的）动作，这就是我们通常说的“算法”。

当今我们的日常生活已经高度依赖信息技术，似乎计算机什么都能干，但稍稍了解计算机内部结构的人都知道“那个机器”本身其实只能做为数不多的基本动作。能够让计算机系统（不仅仅是你能看见的硬件设备）似乎无所不能的关键是各种各样的算法。

人用“智慧”设计算法，能够让机器表现出“智能”。但是我们还不知道如何在可以预见的时间段内让机器具有智慧（尽管我们相信科学技术会以超出我们所有人的想象更迅速地发展）。

前些年开始流行一个词：计算思维。究竟什么是计算思维，计算机工作者至今并没有能给出一个明确的定义。至少我们可以将计算思维理解成人的一种思维方式，它应该能使得计算机更好地（包括通过表现出“智能“的方式）服务于人类。

从当前计算机应用水平来看，算法是计算思维的一种具体体现。人们已经设计出许多非常“聪明“的算法，极大地提高了我们解决问题的能力，但应用中复杂的问题仍然期待我们给出更有效的算法。这是计算机科学家工作的一个重要方面。

任何学科的发展都依赖于一代一代优秀的年轻人的加入，其他学科的经验表明，科学技术普及是引导青少年走向探索未知、创造未来的科学技术道路的重要举措。

坦率地说，计算机科学普及工作与一些传统学科相比差距巨大。一方面是因为计算机科学历史较短。更深层的一个原因应该是认识层面的。信息技术发展迅速，渗透面极广，但社会公众更多的是从技术与实用的角度，而不是从思维方式的角度看待它。现在有非常多的中小学生，甚至幼儿园小朋友在学“编程”。这是很好的事，但只是学会某种语言（不管是C++、Java还是Python）编程只是“技术”，充其量说是“智能”，却绝不是“智慧”。

我曾经听一位钢琴教师说起一位“琴童”的家长问她：“我的孩子通过钢琴十级了，下面该做什么呢”。这位教师的评论多少反映了她对当前社会公众对待学习急功近利态度的不认同。思维能力的培养是一个过程，目的是培养能应对不可预知的问题的基础能力。而应对“题型”，使用“工具”尽管也是学习中不可忽缺的部分，但却不能替代思维能力的培养。

思维方式不是可以用某种“模式”描述的，如何将计算思维培养融入基础教育中是当前一个重要的问题。本书是在一系列给中学生普及计算思维的讲座的基础上整理编写的。选择了计算机科学中一些有重要应用价值的典型问题，但讲述方式充分体现了趣味性。当我第一次看到此书时就被它吸引。

计算机专业是中国工程类专业中规模最为庞大的。时时听到有教师埋怨现在的学生“不好学”。中国传统的教育相对忽视学生的兴趣，强调学习是“责任”，“学习就应该刻苦”。其实兴趣是最好的学习动力，激发学生的兴趣本身也是教师的责任。

本书提供了一个如何提高学生兴趣的很好例子。讨论的每个话题都能体现作者“深入浅出”的用心。每个话题都用日常有趣的问题引入，介绍算法时注意引导读者考虑算法背后的基本思想。但作者又不局限于简单的“问题－答案”描述。尽管是普及读物，作者仍然会在可以理解的范围内引导读者考虑算法的正确性以及复杂性等可以更深入探讨的方面。在若干章节中，作者还通过实验数据来分析结果，注意到体现计算机算法工程性的一面。这甚至做到了当前我们大多数算法课程都没有做到的事。但我特别要强调一点，这都是在不牺牲普及读物的基本可理解性的要求下做到的。尽管本书是多位作者讲稿的合集，但显然出版前还是注意到风格的统一，避免给读者带来阅读上的不适。

广大青少年读者，特别是那些学习过一点编程的中小学生，一定会通过阅读本书在计算思维素养上得到提升。

但绝不仅限于此。本书同样适合已经进入大学计算机类专业学习的学生阅读。他们能够从中欣赏计算机科学有趣的一面，也能对“我们如何解题”有更深入的理解。

我多年来一直从事计算机算法方面的教学，对本书中涉及的内容都很熟悉。但是整个翻译过程仍然让我觉得受益匪浅。我感到尽管本书是针对中学生的普及读物。但是对于从事计算机教学的广大教师也是很值得阅读的。我甚至将其中部分内容用于中国计算机学会每年举办的“计算机教学改革导教班”。我希望计算机专业教师（特别是基础课教师）更关注这样的普及读物，并能从中得到启发。

陈道蓄

2018 年 8 月于南京

前　言

Algorithms Unplugged

最近几十年来许多技术创新和成果都依赖于算法思想，这些成果广泛应用于科学、医药、生产、物流、交通、通信、娱乐等领域。高效的算法使得你的个人电脑得以运行新一代的游戏，这些复杂的游戏在几年前可能都难以想象。更重要的是这些算法为一些重大科学突破提供了基础。例如，人类基因组图谱解码得以实现与新算法的发明是分不开的，这些算法能将计算速度提高几个数量级。

算法告诉计算机如何处理信息，如何执行任务。算法组织数据，使得我们能有效地搜索。如果没有聪明的算法，我们一定会迷失在互联网这个巨大的数据丛林中。同样，如果没有天才的编码和加密算法，我们也不可能在网络上安全地通信。天气预报与气候变化分析也依靠高效模拟算法。工厂生产线和物流系统有大量复杂的优化问题，只有奇巧的算法能帮助我们解决。甚至当你利用 GPS 寻找附近的餐厅或咖啡馆时，也要靠有效的最短路计算才能获得满意的结果。

并非像很多人认为的，只有计算机中才需要算法。在工业机器人、汽车、飞机以及几乎所有家用电器中都包含许多微处理器，它们也都依赖算法才能发挥作用。例如，你的音乐播放器中使用聪明的压缩算法，否则小小的播放器会因为存储量不足而无法使用。现在的汽车和飞机中有成百上千的微处理器，算法能帮助控制引擎，减少能耗，降低污染。它们还能控制制动器和方向盘，提高稳定性与安全性。不久的将来，微处理器可能完全替代人，实现汽车的全自动驾驶。目前的飞机已经能做到在从起飞到降落的全过程中无须人工干预。

算法领域最大的进步都来自美好的思想，它指引我们更有效地解决计算问题。我们面对的问题绝不局限于狭义的算术计算，还有很多表面上不是那么“数学化”的问题。例如：

- 如何走出迷宫？
- 如何分割一张藏宝图让不同的人分别保存，但只有重新拼合才可能找到宝藏？
- 如何规划路径，用最小成本访问多个地方？

这些问题极具挑战，需要逻辑推理、几何与组合想象力，还需要创造力才能解决。这些就是设计算法所需要的主要能力。

本书包括不同作者撰写的 41 篇文章，用非技术化的语言介绍了一些最著名的算法思想。多数文章源自德语大学中发起的科普活动，初衷是让高中生领会算法和计算机科学的奇妙与魅力。阅读本书不需要任何关于算法和计算的预备知识。我们希望不仅学生，而且包括希望了解迷人的算法世界的成年人都能从本书中得到启发与乐趣。

第一部分
Algorithms Unplugged

搜索与排序

Martin Dietzfelbinger，德国伊尔梅瑙工业大学
Christian Scheideler，德国帕德博恩大学

每个孩子都知道排好次序的东西找起来比较容易，至少数量很多时一定如此。经验告诉人们，通过给物体定义次序，我们可以把拥有的东西分门别类地分别放置，而且很容易记住什么东西在什么地方。当然你可以将所有的袜子随意放在某个抽屉里。但如果像 DVD 之类的碟片，要想在许多碟片中很快找到想要的那个，最好还是排好次序。但这里说的“很快”究竟是什么意思呢？我们究竟能多快地排序，又能多快地找到呢？这就是本书第一部分要讨论的重要话题。

第 1 章讨论的是一种称为二分搜索的快速搜索策略。这种策略假设我们已经将要搜寻的所有对象（这里是 CD）排好了序。第 2 章介绍简单的排序策略。其原理是两两比较相邻的对象，需要时就交换它们的位置，直到所有对象排好序。不过当对象数量增大时，这些策略需要的工作量迅速增加，所以它们只有在对象数量不多时才有效。第 3 章介绍两种算法，即使对象数量很多仍然能够较快地实现排序。第 4 章介绍并行排序算法。所谓“并行”是指我们可以同时执行多个比较操作，这样所需的时间显然就会少于逐个进行比较的算法。这样的算法特别适合于有多个处理器（或多核处理器）能够同时执行不同任务的计算机，它们也能用来设计专门用于排序的芯片或者机器。第 5 章是讨论排序的最后一章，介绍拓扑排序的方法。拓扑排序在下面的情况下很有必要：我们有一系列任务需要完成，任务之间存在依赖关系，比如任务 B 只能在任务 A 完成之后开始执行。拓扑排序的目的是给出所有任务顺序执行的列表，不会与存在的依赖关系发生冲突。

第 6 章又回到搜索问题。这次考虑的是文本搜索。具体地说，就是判断在某个文本中是否存在给定的字符串。在文本不是太长、要找的字符串很短的情况

下，人们能很快做出判断，但给计算机设计一个高效的搜索过程就不那么简单了。第 6 章介绍了一种算法，它在实际应用中速度非常快，但在某些特别的情况下也需要很长的搜索时间。

本部分的剩余章节讨论一些看上去差异很大的搜索问题。如何能找到迷宫出口，而且不至于无休止地兜圈子或者反复走同一条路？第 7 章表明如果能沿途设置标记（如同我们在现实中用粉笔画线），我们就能利用一种称为深度优先搜索的基本方法解决这个问题。有趣的是，如果你想系统地搜索万维网的某个部分，甚至你想自己生成一个迷宫，深度优先搜索方法也有效。第 8 章我们再次考虑迷宫问题。不过这次我们只能使用指南针（因此有方向的概念），但不能设置标记。解决这个问题也有非常聪明的算法，称为 Pledge 算法。Pledge 算法还可以用于机器人在不规则地放置了障碍物的平面迷宫中寻找路径。第 9 章介绍的算法是深度优先搜索的特殊应用，能够在迷宫、街道网络或者社会关系网络中找出回路。有时候识别是否有回路非常重要，它有助于解决死锁问题。例如有时候有依赖关系的任务之间形成相互等待的状态，结果谁也动不了，这被称为“死锁”。令人惊奇的是有非常简单而聪明的算法能发现网络中所有可能存在的回路。

第 10 章是第一部分的最后一章，讨论的是万维网的搜索引擎。这里的场景是用户提出搜索要求，期待搜索引擎返回尽可能与搜索要求相关的网页链接的列表。这可不是简单的任务，可能有数千甚至数十万网页中包含用户提出的关键字，重要的是搜索引擎需要确定其中哪些是与用户要求最相关的。那么搜索引擎究竟该怎么做呢？第 10 章解释了其基本原理。

第 1 章
Algorithms Unplugged

二分搜索

Thomas Seidl，德国亚琛工业大学
Jost Enderle，德国亚琛工业大学

我新买的 Nelly 的唱片哪儿去啦？我那专横的妹妹 Linda 有整洁癖，肯定是她将唱片又插进唱片架上了。我告诉她新买的唱片别插上去。这下我得在架子上的 500 张唱片中一张一张地找了，这该找到什么时候啊！

不过，走运的话也可能并不需要查看所有唱片就碰到了。但最坏的情况是 Linda 又把唱片借给朋友了，那得查完所有唱片才知道不在这里了，然后只好去听广播啦。

找找看吧！ Aaliyah，AC/DC，Alicia Keys……嗯，Linda 好像按字母顺序给唱片排过序了。这样的话我找 Nelly 的唱片就容易多了。我先在中间试试。Kelly Family，这太偏左了，必须往右边找。Rachmaninov，这又太偏右了，再往左一点儿……Lionel Hampton，右了点儿，但不远了。Nancy Sinatra……Nelly，找到啦！

这倒很快！因为唱片已经排了序，我只要来回跳几次就找到目标了！即使我要的唱片不在架子上，我也能很快发现。不过如果唱片很多，比如说 10 000 张，那可能得来回跳上几百次吧。我很想知道如何计算次数。图 1-1 给出了不同搜索方法的示意。

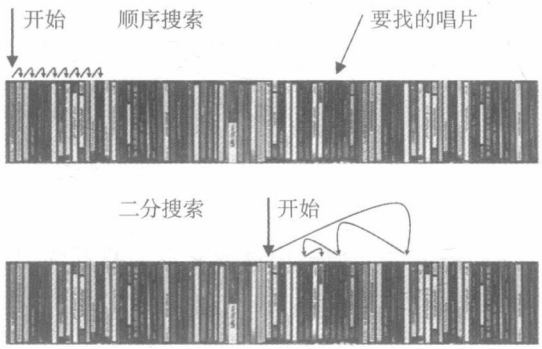


图 1-1　顺序搜索与二分搜索图示

1.1　顺序搜索

Linda 从去年开始学习计算机科学；她应该有些书能告诉我答案。我看看，“搜索算法”可能有用。这里说了如何在一个给定集合（这里是唱片）中按照关键字（这里用艺术家的名字）找一个对象。我刚才的做法应该是“顺序搜索”，又叫“线性搜索”。就像我想的一样，

为了找一个关键字，平均得检查一半的唱片。搜索的步数和唱片数成正比，换句话说，唱片数增加一倍，搜索时间也就增加一倍。

1.2　二分搜索

我用的第二种技术好像有个特别的名字，叫“二分搜索”。给定要找的关键字以及排好次序的对象列表，搜索从中间那个对象开始，和关键字进行比较。如何中间那个对象就是要找的，搜索就结束了。否则，按照要找的关键字是小于还是大于当前检查的对象决定该向左还是该向右继续搜索。接下来就是重复上面的过程。如果找到了搜寻的对象，或者当前可能搜索的区间已经不能再切分了（也就是说如果表中有要找的对象，当前位置就该是目标应该在的位置），搜索就终止。我妹妹的书中有相应的程序代码。

在这段代码中，`A`表示一个“数组”，也就是由带编号的对象（我们称其为数组的元素）构成的数据列表，编号就像唱片在架子上的位置。例如，数组中第5个元素写为`A[5]`。如果我们的架子上放了500张唱片，我们要找的关键字是`"Nelly"`，那就得调用`BINARYSEARCH(rack, "Nelly", 1, 500)`搜索要找的唱片所在位置。程序执行时，开始的`left`值为251，`right`值为375，以此类推。

函数`BINARYSEARCH`返回的值是`key`在数组`A`中的位置，位置介于当前的`left`和`right`之间。

```
1  function BINARYSEARCH (A, key, left, right)
2  while left ≤ right do
3      middle := (left + right)/2      // 求 middle，取整数值
4      if A[middle] = key then return middle
5      if A[middle] > key then right := middle − 1
6      if A[middle] < key then left := middle + 1
7  endwhile
8  return “未发现”
```

1.3　递归实现

在Linda的书中还有另外一个二分搜索算法。同样的功能为什么需要不同算法呢？书上说第二种算法采用“递归方法”，那又是什么呢？

我再仔细看看……“递归函数是一种利用自身来定义或者调用自己的函数。”求和函数sum就是个例子。函数sum的定义如下：

$$\text{sum}(n) = 1 + 2 + \cdots + n$$

也就是前n个自然数相加，所以，当$n=4$，可得：

$$\text{sum}(4) = 1 + 2 + 3 + 4 = 10$$

如果我们想计算对于某个n的sum函数值，而且已经知道对于$n-1$的函数值，那只要再加上n就可以了：

$$\text{sum}(n) = \text{sum}(n-1) + n$$

这样的定义式就称为“递归步”。当要计算对于某个n的sum函数值时，我们还需要一个最小的n对应的函数值，这称为奠基：

$$\text{sum}(1) = 1$$

按照递归定义，我们现在计算 sum 函数值的过程如下：

$$\begin{aligned}\text{sum}(4) &= \text{sum}(3) + 4\\ &= (\text{sum}(2) + 3) + 4\\ &= ((\text{sum}(1) + 2) + 3) + 4\\ &= ((1 + 2) + 3) + 4\\ &= 10\end{aligned}$$

二分搜索的递归定义是一样的：函数在函数体中调用自己，而不是反复执行一组操作（那称为循环实现）。

函数 BINSEARCHRECURSIVE 返回的值是 key 在数组 A 中的位置，位置介于当前的 left 和 right 之间

```
1  function BINSEARCHRECURSIVE(A, key, left, right)
2  if left > right return not found
3  middle := (left + right)/2      // 找到介于中间的值
4  if A[middle] = key then return middle
5  if A[middle] > key then
6     return BINSEARCHRECURSIVE(A, key, left, middle − 1)
7  if A[middle] < key then
8     return BINSEARCHRECURSIVE(A, key, middle + 1, right)
```

和前面一样，A 是要搜索的数组，key 是要找的关键字，left 和 right 分别是搜索区域的左右边界。如果我们要在包含 500 个元素的数组 rack 中找 Nelly，我们采用类似的函数调用 BINSEARCHRECURSIVE(rack,"Nelly",1,500)。但这里不再通过程序循环使得搜索区域左右边界逐步靠近，而是直接修改边界值执行递归调用。实际执行的递归调用序列如下：

```
BINSEARCHRECURSIVE (rack, “Nelly” , 1, 500)
BINSEARCHRECURSIVE (rack, “Nelly” , 251, 500)
BINSEARCHRECURSIVE (rack, “Nelly” , 251, 374)
BINSEARCHRECURSIVE (rack, “Nelly” , 313, 374)
BINSEARCHRECURSIVE (rack, “Nelly” , 344, 374)
...
```

1.4　搜索的步数

至此，我们仍然不知道要找到所需的对象究竟该执行多少搜索步。如果运气好，一步就能找到。反之，如果要找的对象不存在，我们必须来回跳动直至对象应该处于的位置。这样就需要考虑究竟数组能够被切为两半多少次，或者反过来说，当执行了一定数量的比较操作后，究竟多少元素可以被确定是或者不是目标对象。假设要找的对象确实在表中，一次比较可以确定 2 个元素，两次比较可以确定 4 个元素，三次比较就能确定 8 个元素。因此执行 k 次比较操作能够确定 $2 \cdot 2 \cdot \cdots \cdot 2$（$k$ 次）$= 2^k$ 个元素。由此可知 10 次比较可以确定 1024 个元素，20 次比较能确定的元素超过 100 万个，而 30 次比较能确定的元素多达 10 亿个以上。如果目标对象不在数组中，则需要多比较一次。为了能根据元素个数确定比较次数，我们需

要逆运算，也就是 2 的乘幂的反函数，即“以 2 为底的对数”，记作 $\log_2$。一般地说：

$$\text{假设 } a = b^x, \text{ 则 } x = \log_b a \tag{1-1}$$

对于以 2 为底的对数，$b = 2$：

$$\begin{aligned}
2^0 &= 1, & \log_2 1 &= 0\\
2^1 &= 2, & \log_2 2 &= 1\\
2^2 &= 4, & \log_2 4 &= 2\\
2^3 &= 8, & \log_2 8 &= 3\\
&\vdots & &\vdots\\
2^{10} &= 1\,024, & \log_2 1\,024 &= 10\\
&\vdots & &\vdots\\
2^{13} &= 8\,192, & \log_2 8\,192 &= 13\\
2^{14} &= 16\,384, & \log_2 16\,384 &= 14\\
&\vdots & &\vdots\\
2^{20} &= 1\,048\,576, & \log_2 1\,048\,576 &= 20
\end{aligned}$$

因此，若 k 次比较操作能确定 N（$= 2^k$）个元素，那么对于含 N 个元素的数组，二分搜索需要执行 $\log_2 N = k$ 次比较操作。如果我们的架子上放了 10 000 张唱片，我们需要比较 $\log_2 10\,000 \approx 13.29$ 次。因为不可能比较“半次”，需要的次数为 14。要想进一步减少二分搜索需要的步数，可以在搜索过程中不是简单地选择中间元素进行比较，而是尝试在搜索区域内更准确地“猜测”可能的位置。假设在已排序的唱片中搜索的对象名按字母顺序更靠近区域开始处，例如找 Eminem，显然选择前部的某个位置进行比较更好些。反之，要找“Roy Black”，从靠后的地方开始更合理。若要更好地改进，就得考虑每个字母可能出现的频率，例如首字母是 D 或 S 的艺术家通常比首字母是 X 或 Y 的更常见。

1.5　猜数游戏

今晚我要考考 Linda，让她猜 1 到 1000 之间的某个数。只要上课没睡觉，她就应该能最多通过 10 个“是 / 否”的问题得到结果。（图 1-2 显示如何只问 4 个问题就猜出 1 到 16 之间的某个数。）

为了避免反复问那些“是小于某个数吗？”或者“是大于某个数吗？”那样乏味的问题，我们可以选择问“是奇数吗？”或“是偶数吗？”。因为一个回答就可以让我们排除一半的可能性。类似的问题包括“十（百）位数是奇（偶）数吗？”，像这样的问题同样可以使搜索空间（大致）缩小一半。不过要确认考虑了所有可能的数，我们还得回到通常采用的减半方法（那些已经被排除的数实际上已考虑在内）。

如果采用二进制表示数，这个过程甚至会更简单。十进制系统是用“10 的乘幂的和”的形式表示数，例如：

$$\begin{aligned}
107 &= \mathbf{1} \cdot 10^2 + \mathbf{0} \cdot 10^1 + \mathbf{7} \cdot 10^0\\
&= \mathbf{1} \cdot 100 + \mathbf{0} \cdot 10 + \mathbf{7} \cdot 1
\end{aligned}$$

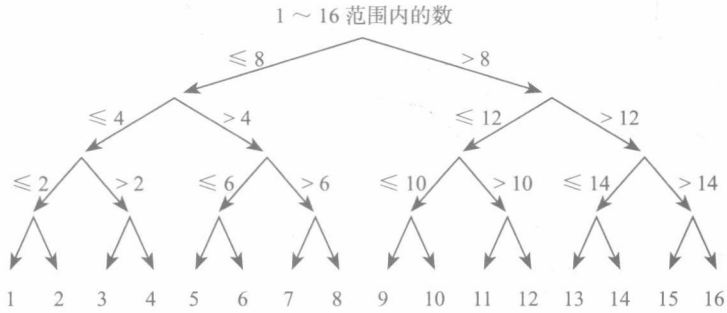


图 1-2　在 1 ～ 16 范围内猜出某个数的图示

而在二进制系统中数是用“2 的乘幂的和”的形式表示的：

$$107 = \mathbf{1} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$$

$$= \mathbf{1} \cdot 64 + \mathbf{1} \cdot 32 + \mathbf{0} \cdot 16 + \mathbf{1} \cdot 8 + \mathbf{0} \cdot 4 + \mathbf{1} \cdot 2 + \mathbf{1} \cdot 1$$

因此 107 的二进制表示为 1101011。要猜出一个二进制表示的数只要知道它最多多少位就足够了。位数用以 2 为底的对数很容易计算。如果猜一个 1 到 1000 之间的数，可以计算如下：

$$\log_2 1000 \approx 9.97 \text{（向上取整）}$$

也就是说共有 10 位。因此问 10 个问题足够了：“第 1 位数是 1 吗?”“第 2 位数是 1 吗?”“第 3 位数是 1 吗?”等等。最后所有位都知道了还必须转换为十进制数，用一个掌上的计算器就能解决了。

第 2 章

Algorithms Unplugged

插入排序

Wolfgang P. Kowalk，德国奥尔登堡大学

我们要把书架上所有的书按照书名排序，这样需要哪本书时很快就能找到。

如何快速地实现排序呢？我们可以有几种不同的想法。例如我们可以依次查看每本书，一旦发现两本紧挨着的书的次序不对就交换一下位置。这种想法能行，因为最终任何两本书的先后都不会错，但这平均要花费太长的时间。另一种想法是先找出书名最“小”的那本书放在第一个位置，然后在剩下的书中再找出最“小”的放在紧挨着的后面位置，以此类推直到所有书都放在了正确的地方。这种想法也能行；但是由于大量有用的线索没有利用，多花费了许多时间。下面我们试试其他的想法。

下面的想法似乎比上面讨论的更加自然。第一本书自然是排好的。接下来我们拿第一本书的书名与第二本书的书名做比较，如果次序不对就交换两本书的位置。然后我们看下一本书在前面已经排好序的部分中应该放在什么位置。这可以反复进行直到为所有的书安排了正确的位置。因为前面的书排序时提供的信息可供后面使用，这个方法应该效率高一些。

现在把这个算法再细细看一下。第一本书单独考虑可以看作排好了序。我们假设当前考虑的书是第 i 本书，而它左边所有的书都已排好序了。要将第 i 本书加入序列中，我们首先查找它正确的位置，随后将书插入即可；为此要将在正确位置右边的所有书向右移动一个位置。接下来对第 $i+1$ 本书重复以上过程，以此类推直到所有的书放到了正确位置。这个方法能快速产生正确结果，特别是如果我们采用第 1 章介绍的二分搜索寻找正确插入位置则效果更明显。

我们现在来看看对任意数量的书，这个直观的方法如何实现。为了描述起来简单一些，我们用数字代替书名。

图 2-1 中左边的 5 本书（1，6，7，9，11）已经排好序，而书名为 5 的书位置不正确。为了将 5 放入正确位置，首先与 11 交换位置，再与 9 交换位置，以此类推直到 5 到达正确位置。然后我们再处理书名为 3 的书，同样通过与左侧的书交换来到达正确位置。显然最终所有的书都会放到正确的地方（见图 2-2）。

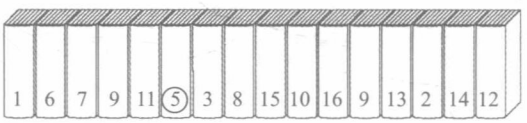


图 2-1　前 5 本书已排好序