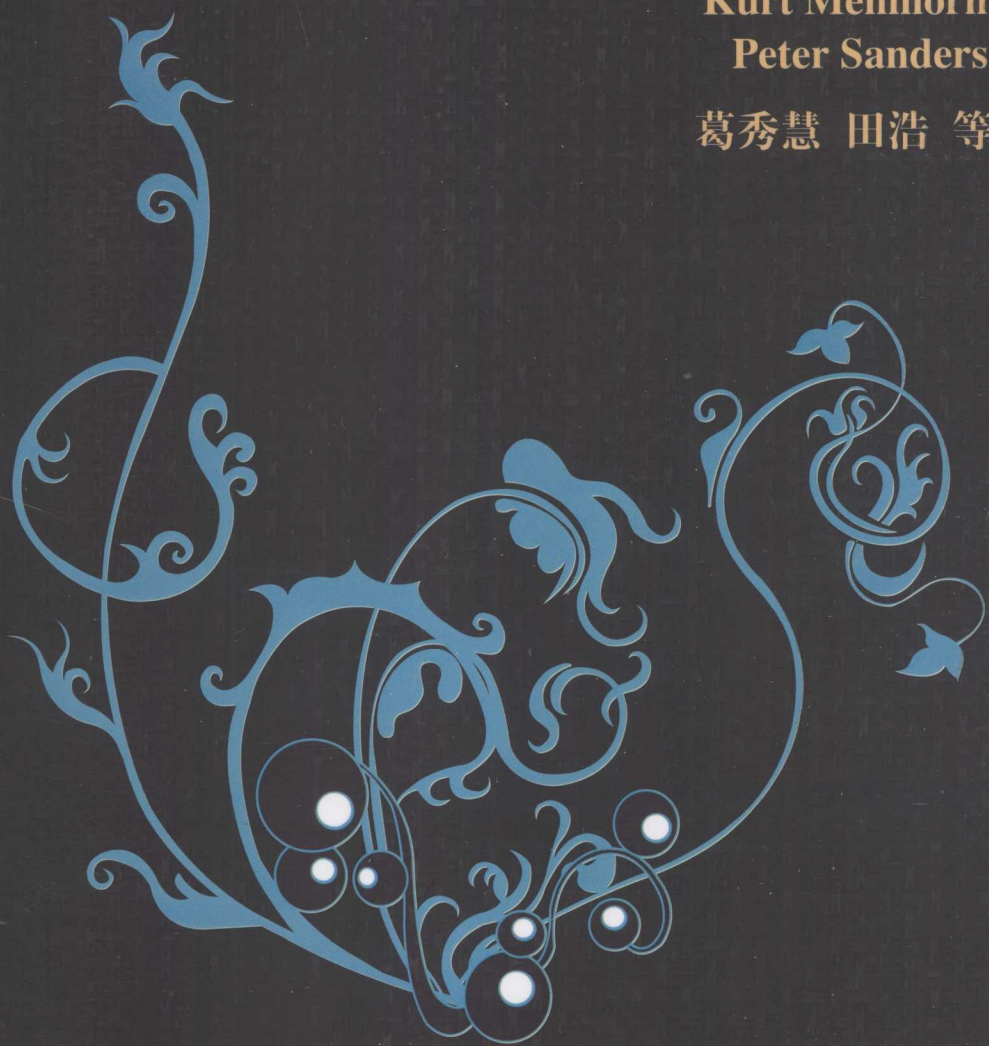
Springer

世界著名计算机教材精选

算法与数据结构

Kurt Mehlhorn
Peter Sanders 著

葛秀慧 田浩 等 译



ALGORITHMS AND DATA STRUCTURES

THE BASIC TOOLBOX

清华大学出版社

世界著名计算机教材精选

算法与数据结构

Kurt Mehlhorn
Peter Sanders
著
葛秀慧　田　浩　等译

清华大学出版社
北　京

本书为英文版 *Algorithms and Data Structures: The Basic Toolbox* 的简体中文翻译版，作者 **Kurt Mehlhorn, Peter Sanders**，由 **Springer** 出版社授权清华大学出版社出版发行。

北京市版权局著作权合同登记号　图字：01-2011-7558 号

图书在版编目（CIP）数据

算法与数据结构/（德）梅霍内（Mehlhorn, K.），（德）桑德斯（Sanders, P.）著；葛秀慧等译. —北京：清华大学出版社，2013.4
（世界著名计算机教材精选）
ISBN 978-7-302-31017-4

Ⅰ. ①算… Ⅱ. ①梅… ②桑… ③葛… Ⅲ. ①算法分析－教材 ②数据结构－教材 Ⅳ. ①TP301.6 ②TP311.12

中国版本图书馆 CIP 数据核字（2012）第 304401 号

责任编辑：龙启铭
封面设计：傅瑞学
责任校对：梁　毅
责任印制：杨　艳

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
课 件 下 载：http://www.tup.com.cn, 010-62795954

印 装 者：北京市清华园胶印厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：14.75　　**字　　数**：369 千字
版　　次：2013 年 4 月第 1 版　　**印　　次**：2013 年 4 月第 1 次印刷
印　　数：1～3000
定　　价：29.50 元

产品编号：038328-01

译 者 序

每翻译一本书，都在面对一件自己的作品，在内心深处都想让这件作品完美。在拿到本书时，一读第 1 章，就被作者有趣的比喻所吸引。它的标题竟然是“开胃菜”。细读下去，被作者严谨求实的态度所震撼。再读下去，又被作者丰富的经验和精练的语言所吸引。作者从日常生活中的实际问题开始，娓娓道出简单而直接的解决方法，然后加以引申，归纳为算法中的某种数据结构，最后给出算法的具体实现细节，可谓用心良苦。不愧为一本难得的经典之作。

本书共分 12 章，涵盖了数据结构的数组与链表、散列表与关联数组、排序与选择、优先队列、有序序列、图的表示、图的遍历、最短路径、最小生成树与优化。第 1 章作为一个引子，作者以读者熟悉的整数乘法为核心，介绍了大数乘法算法，以此激发读者对算法的兴趣。第 2 章介绍了本书算法所需的基础知识——渐近表示法、术语、机器模型、高级伪代码表、复杂度、平均情况分析、随机算法、图的基础、复杂性类 P 和 NP，同时还给出了本书的第一个综合性示例——有序数组的二分查找。第 3～11 章是数据结构课程必须学习的内容，其与其他教科书的不同之处在于：作者独具匠心的从问题域到解域的思考方法，这种学习思想是非常棒的。在第 12 章中，以背包问题为主线，介绍了 7 种遗传方法：黑盒求解器、贪婪算法、线性规划、动态规划、系统搜索、局部搜索和进化算法。特别是局部搜索算法中的爬山、模拟退火和图着色使人印象深刻。

本书的习题也非常值得一提，它改变了以往习题集中在每章之后的惯例，而是在讲完有关问题之后，就有对应的习题。习题精练，深度适中，是正文内容的拓展与延续。使读者能进一步深入思考所学的相关的知识。

在涉及更高级的数据结构与算法主题时，本书是逐步地引入更多的数学处理，包括定理和证明。以这种方法，为读者提供更专业的数学知识。

为了使读者专注于算法的思想，而不是编程的细节，本书运用了图片、文字和高级伪代码来解释算法，同时，还将抽象的思想链接到简洁而高效的 C++ 和 Java 的真实实现。

在翻译过程中，虽然力求完善，经过了初翻、再翻和精翻，但由于作者水平有限，难免有所纰漏，希望广大同行、专家和读者给予批评指正。另外，田浩、刘展威、张桂香、王顶、王春海、刘秋红、刘朝晖、焦仁普、朱书敏、盖俊飞、李超、郭立甫和张小蕊等参加了本书的翻译，在此一并表示谢意。

译 者

前　言

算法是每个复杂计算机应用的核心。因此，每位计算机科学家和专业程序员都应该熟悉算法的基本工具包：即熟悉并能有效组织和检索数据的结构、常用的算法，以及用于建模、理解和求解算法问题的通用技术。

本书简练地介绍了这些基本工具包，旨在使学生和专业人员熟悉编程语言和基本的数学知识。在本科生的算法课程中，我们使用了本教材。在研究生课程中，将本书的大部分内容作为预备知识，然后精讲带星号的章节和更高级的材料。我们相信，只要教材包含了大量的示例、图片、通俗的解释、练习以及与真实世界的链接，即使是本科生，也能完全掌握这些简练而明晰的知识。

每章的基本结构大致相同，都从讨论现实生活中出现的问题开始。我们阐明最重要的应用，尽可能通俗地、按需要正式地介绍简单的解决方案，使读者能真正地理解当前问题。当需要阐明更高级的可选问题时，会逐步引入更多的数学处理，包括定理和证明。以这种方法，为读者提供更多的数学专业知识。书中还有更高级的章节（标有 * 号），建议读者在初次阅读时，跳过这些章节。练习提供了更多示例和可选方法，读者通过练习能再次分析与思考相关问题。所以，我们极力建议，在你初读本书时，即使没时间做练习，也要仔细地分析与思考这些练习。为了能专注于算法的思想，而不是编程细节，本书使用了图片、文字和高级伪代码来解释算法。在“实现提示”小节中，将抽象的思想链接到简洁而高效的实现，这些实现是都由 C++ 和 Java 等真正的编程语言编写的。每章的最后一节都给出了进一步的读物，简要地介绍了算法的研究现状、概述和高级解决方案。

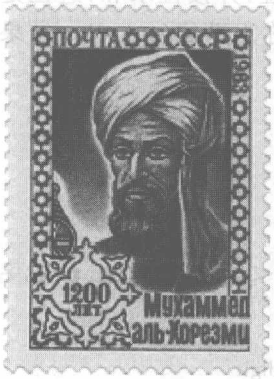
在计算机科学中，即使处于基本工具包这一层次，算法也是一个崭新、活跃的领域。我们用最新的方法来介绍算法，如明确介绍了不变量。我们还讨论了最新的趋势，如算法工程、存储器的层次结构、算法库和算法证明。

我们选择以问题域而不是以解技术来组织本书的大部分材料（最后一章的优化技术除外）。我们发现通过问题域来介绍算法，能更简洁地表示算法。但是，使读者和学生更好地掌握可用技术也非常重要。因此，最后一章是以技术来组织结构。

Kurt Mehlhorn
Peter Sanders

目　录

第1章　开胃菜：整数运算



开胃菜是为了刺激食欲，在用餐时上的第一道菜。本章就像用餐时吃的开胃菜一样。通过为你展示令人惊奇的结果，激发你对算法[①]技术的兴趣。学校所学的整数乘法并不是最好的乘法算法；还有许多更快的算法来计算大整数（大整数一般有上百位甚至上万位）乘法，在本章中，我们会教你其中一种算法。

大整数运算用于许多领域，如加密、几何计算、计算机代数等。改进乘法算法不仅是一种智力创新，对应用程序而言，也是非常重要的。一方面，在简单的应用中，我们就能学习基本的分析和基本的算法工程技术。另一方面，我们还能理解理论与实验之间的相互作用。

假设，用数字串表示整数。在基为B（其中B是大于某个数的整数）的数制中，包含数字0,1,…,B−1，数字串 $a_{n-1}a_{n-2}\cdots a_1a_0$ 表示数 $\sum 0\leqslant i<n^{a_i B^I}$。最重要的数制一般都使用小值的基B，如，基为2，包含数字0和1；基为10，包含数字0～9；基为16，包含数字0～15（经常写作0,…,9,A,B,C,D,E,F）。较大的基（如 2^8、2^{16}、2^{32} 和 2^{64}）也是非常有用的。例如，以2为基，“10101”的表示为：$1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0=21$，以10为基，924的表示为：$9\times10^2+2\times10^1+4\times10^0=924$。

假设在处理中，使用两种原语操作：三个数相加得到一个两位数（有时将其称为全加器）；两个数相乘得到一个两位数[②]。例如，以10为基，我们有

$$\begin{array}{r}3\\5\\5\\\hline13\end{array}\quad\text{以及}\quad 6\times7=42$$

可以通过算法所执行的原语操作次数来测量算法的效率。

对于任意m≥n的整数，通过添加额外的前导零，都能将n位整数变成m位整数。具体地说，425和000425表示相同的整数。我们使用a和b作为加法或乘法的两个操作数，并假设在本节中，a和b是n位整数，且两个操作数具有相同的长度，这样无须改变本章的重要信息就能简化表示。在本章最后，仍会使用这一假设。数a是指 $a_{n-1}\cdots a_0$，a_{n-1} 是最高有效位（也称前导位），a_0 是最低有效位，记作 $a=(a_{n-1}\cdots a_0)$。同样，用 $b_{n-1}\cdots b_0$ 来表示数b，记作：$b=(b_{n-1}\cdots b_0)$。

---

① 本页的前苏联邮票上是Muhammad ibn Musa al-Khwarizmi（约生于780年；卒于835到850年之间），波斯数学家和天文学家，出生地是Khorasan省，现今称为乌兹别克斯坦（Uzbekistan）。“算法”一词源于该数学家的名字。

② 观察三个数的和最多是 $3(B-1)$，两个数的积最多是 $(B-1)^2$，两个表达式的上界都是 $(B-1)\cdot B^1+(B-1)\cdot B^0=B^2-1$，该值是两个数能表示的最大整数。

1.1 加 法

我们都知道如何将两个整数 $a=(a_{n-1}\cdots a_0)$ 和 $b=(b_{n-1}\cdots b_0)$ 相加。只需先写出一个整数，然后在其下写出另一个整数，书写时，对齐两个整数的最低有效位，并按位求和，进位是一个单独的数，它从低位向相邻的高位进位。我们将这个单独的数称为**进位**(carry)。结果得到一个 $n+1$ 位的整数 $s=(s_n\cdots s_0)$。图示如下：

$$\begin{array}{rl} a_{n-1}\cdots a_1\ a_0 & \text{第 1 个操作数} \\ b_{n-1}\cdots b_1\ b_0 & \text{第 2 个操作数} \\ \underline{c_n\ c_{n-1}\cdots c_1\ 0} & \underline{\text{进位}} \\ s_n\ s_{n-1}\cdots s_1\ s_0 & \text{和} \end{array}$$

其中 c_n 到 c_0 是进位序列，$s=(s_n\cdots s_0)$ 是和。令 $c_0=0$，则对于任意 $0\leqslant i<n$，都有 $c_{i+1}\cdot B+s_i=a_i+b_i+c_i$，且 $s_n=c_n$。对此进行编程，程序可以写为：

```
c=0: Digit                                    //Variable for the carry digit
for i:=0 to n-1 do add a_i, b_i, and c to form s_i and a new carry c
s_n:=c
```

对于每一位都需要一次原语操作，因此，总计为 n 次原语操作。

定理 1.1 *两个 n 位整数相加，只需要 n 次原语操作。结果是一个 n+1 位的整数。*

1.2 乘法：学校方法

本节先回顾一下“学校方法”，在稍后的章节中再介绍一种更快的大整数计算算法。

我们采用循序渐进的方式进行介绍。先回顾一下如何将 n 位整数 a 乘以 1 位整数 b_j。之所以用 b_j 表示 1 位整数，是因为后面需要这种表示方法。对于 a 的任意数位 a_i，可以产生积 $a_i\cdot b_j$，结果是一个两位整数 $(c_i d_i)$，即

$$a_i\cdot b_j=c_i\cdot B+d_i$$

根据 c 和 d 的数位，可以产生两个整数 $c=(c_{n-1}\cdots c_0 0)$ 和 $d=(d_{n-1}\cdots d_0)$。因为 c 是积中的高阶位，在其后添加了一个零。将 c 和 d 相加，得到积 $p_j=a\cdot b_j$。图示为：

$$(a_{n-1}\cdots a_i\cdots a_0)\cdot b_j\rightarrow \begin{array}{c} c_{n-1}\ c_{n-2}\cdots\ c_i\ c_{i-1}\cdots\ c_0\ 0 \\ \underline{d_{n-1}\cdots\ d_{i+1}\ d_i\cdots\ d_1\ d_0} \\ \text{c 和 d 的和} \end{array}$$

下面，让我们确定原语操作的次数。对于每个 i，为了要产生积 $a_i\cdot b_j$，都需要一次原语操作，合计为 n 次原语操作。然后，将两个 $n+1$ 位数相加，也需要 $n+1$ 次原语操作。因此，总的原语操作次数为 $2n+1$。

引理 1.2 *n 位数乘以 1 位数，需要 2n+1 次原语操作。结果是 n+1 位数。*

当 n 位数乘以 1 位数时，过程还可以略有不同。在一步内，我们就能完成产生积 $a_i\cdot b_j$ 以及对 c 和 d 的求和[①]。即，一次生成最后相加阶段所需的数位 c 和 d。我们之所以选择在

① 在文献的编译器结构和性能优化中，将这种变换称为**循环合并**(loop fusion)。

一步内得到 c 和 d，是因为这样做能简化算法的描述。

练习 1.1　编写一个程序，在一步内完成 a 和 b_j 的相乘。

现在，转到两个 n 位整数的乘法。学校所学的整数乘法计算过程如下：先通过 a 乘以 b 的第 j 位 b_j 产生部分积 P_j，然后，对适当对齐的积 $p_j \cdot B^j$ 求和，得到 a 和 b 的积。图示为：

$$\begin{array}{rrrrrrrrrrr} & & & & & & p_{0,n} & p_{0,n-1} & \cdots & p_{0,2} & p_{0,1}\ p_{0,0} \\ & & & & & p_{1,n} & p_{1,n-1} & p_{1,n-2} & \cdots & p_{1,1} & p_{1,0} \\ & & & & p_{2,n} & p_{2,n-1} & p_{2,n-2} & p_{2,n-3} & \cdots & p_{2,0} & \\ & & & & & \vdots & & & & & \\ \hline & & & p_{n-1,n} & \cdots & p_{n-1,3} & p_{n-1,2} & p_{n-1,1} & p_{n-1,0} & & \end{array}$$

n 个部分积的和

伪代码的描述更为紧凑。先将积 p 初始化为零，然后累加部分积 $a \cdot b_j \cdot B^j$：

```
p=0:N
for j:=0 to n-1 do p:=p+a·b_j·B^j
```

让我们来分析学校方法所需的原语操作次数。产生部分积 P_j 需要 $2n+1$ 次原语操作，因此，产生所有部分积一共需要 $2n^2+1$ 次原语操作。积 $a \cdot b$ 是 2n 位数，因此，对所有 $p+a \cdot b_j+B^j$ 求和就是对 2n 位整数求和。每次这类加法最多需要 2n 次原语操作，这样，所有加法一共需要 $2n^2$ 次原语操作。因此，所需的总原语操作次数不会超过 $4n^2+n$。

通过简单的观察，我们就能改进这个界。数 $a \cdot b_j \cdot B^j$ 是 $n+1+j$ 位，最后一位 j 是零。因此，加法可以从第 $j+1$ 位置开始。此外，当 p 不断累加 $a \cdot b_j \cdot B^j$ 时，令 $p=a \cdot (b_{j-1} \cdots b_0)$，即 p 是 $n+j$ 位。因此，p 和 $a \cdot b_j \cdot B^j$ 的相加等同于两个 $n+1$ 位数相加，所以只需要 $n+1$ 次原语操作。因此，所有加法所需的原语操作次数一共只有 n^2+n。因此，需要证明如下结果。

定理 1.3　*用学校方法将两个 n 位整数相乘，需要 $3n^2+2n$ 次原语操作。*

至此，我们已经分析了整数加法和整数乘法（学校方法）所需的原语操作次数。整数乘法（学校方法）的原语操作次数 M_n 是 $3n^2+2n$。通过观察 $3n^2+2n=n^2(3+2/n)$ 可知，对于大 n，$3n^2+2n$ 基本与 $3n^2$ 相同。所以，我们说 M_n 是**平方增长**（grows quadratically）。通过观察还知道：

$$M_n/M_{n/2}=\frac{3n^2+2n}{3(n/2)^2+2(n/2)}=\frac{n^2(3+2/n)}{(n/2)^2(3+4/n)}=4 \cdot \frac{3n+2}{3n+4} \approx 4$$

即，当实例大小翻倍时，平方增长会使原语操作次数翻两番。

现在假设，我们使用自己所喜爱的编程语言（稍后在本章完成），真正实现了乘法算法，然后，在自己喜爱的计算机上，用不同的 n 位整数 a 和 b 以及不同的 n 运行程序，观察程序运行的时间。我们期待什么样的结果呢？我们需要讨论，以便进一步理解平方增长。原因在于原语操作的次数代表算法的运行时间。先分析两个 n 位整数的加法。当执行程序时，会发生什么呢？对于每个位置 i，数 a_i 和 b_i 都必须移入处理单元，产生和 a_i+b_i+c，结果 s_i 要存储在内存中。进位 c 被更新，索引 i 在递增，且要执行对循环退出的测试。因此，对于每个 i，机器执行的循环次数都是相同的。将每个 i 记作一次原语操作，则原语操作次数就是机器执行循环的次数。当然，一些其他的操作也会影响运行时间。如，使用管道和复杂的传输机制在内存和处理单元之间传输数据，会需要时间，但对所有的 i，产生的影响是相同

的。因此，原语操作的次数能代表在实际机器上的实际实现的运行时间。这一论点可以扩展到乘法，因为一个数乘以一位数与加法的处理过程相似，乘法(学校方法)的第二步等同于一系列的加法。

下面，让我们通过实验来确认上述论点。图 1.1 给出了 C++ 实现的执行时间(学校方法)。在 1.7 节可以找到该程序。对于每个 n，执行大量[①] n 位随机整数的乘法；然后确定平均运行时间 T_n；在表的第 2 列给出了 T_n。表的第 3 列给出了 $T_n/T_{n/2}$ 的比值。图 1.1 也给出了数据点[②]$(\log n, \log T_n)$的标绘图。数据呈现近似的平方增长，与我们通过各种方式得出的推断相同。$T_n/T_{n/2}$ 的比值始终接近 4，双对数图也说明平方增长实质上是一条斜率为 2 的直线。实验结果非常令人鼓舞。我们的理论分析有预测值。我们的理论分析表明原语操作次数是平方增长，在上述讨论中，我们论证了运行时间应与原语操作次数相关，且实际运行时间基本上是平方增长。但是，我们还要清楚存在系统偏差。对于小 n，一行到另一行的增长会小于因数 4，因为是线性关系，所以在运行时间中，常数项仍起到很大的作用。对于大 n，比值非常接近 4。对于非常大的 n(过大则不方便定时)，因为访问内存的时间取决于数据的大小，所以因数可能大于 4。在 2.2 节，我们将讨论与此相关的内容。

n	T_n (sec)	$T_n/T_{n/2}$
8	0.00000469	
16	0.0000154	3.28527
32	0.0000567	3.67967
64	0.000222	3.91413
128	0.000860	3.87532
256	0.00347	4.03819
512	0.0138	3.98466
1024	0.0547	3.95623
2048	0.220	4.01923
4096	0.880	4
8192	3.53	4.01136
16384	14.2	4.01416
32768	56.7	4.00212
65536	227	4.00635
131072	910	4.00449

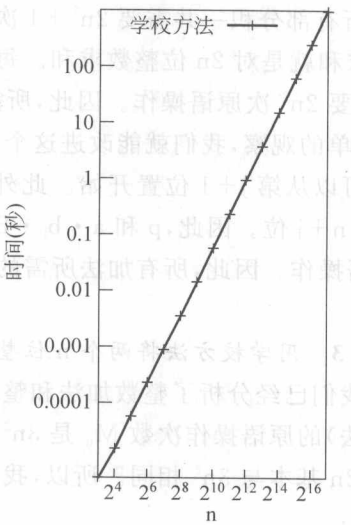


图 1.1 n 位整数乘法(学校方法)的运行时间。在左边，表的三列分别是给定的 n，1.7 节给定的 C++ 实现的运行时间 T_n 和 $T_n/T_{n/2}$ 的比值。右图给出了 $\log T_n$ 与 $\log n$，我们看到的基本是一条直线。进行观察，如果对于某些常量 α 和 β，$T_n = \alpha n^\beta$，则 $T_n/T_{n/2} = 2^\beta$，且 $\log T_n = \beta \log n + \log \alpha$，即 $\log T_n$ 与 $\log n$ 呈线性关系，斜率为 β。在我们的示例中，斜率是 2。请用尺子来检查

练习 1.2 为大整数的加法和乘法编写程序。整数用十进制数序列表示(用编程语言所提供的数组、表或任何数据结构)，并使用内置运算来实现原语操作。然后，编写 ADD、MULTIPLY1 和 MULTIPLY 函数，分别实现整数相加、整数乘以一位整数和整数相乘。

① 测量 CPU 时间的内部时钟以某一单位返回计时，一般单位时间使用毫秒，因此，四舍五入会引入此单位时间一半的误差。所以，为了减少四舍五入产生的影响，实验的定时一定要比单位时间长，这一点非常重要。

② 在整本书中，我们使用 $\log x$ 来表示以 2 为基数的对数 $\log_2 x$。

使用自己的实现来产生自我版的图 1.1。实验使用大于 10 的基，如使用基 2^{16}。

练习 1.3　描述与分析学校方法的除法。

1.3 结果检查

加法和乘法算法都非常简单，因此，会理所应当地认为，使用自己所选的编程语言，就能正确地实现这两种算法。但是，编写软件[①]是非常容易出错的，因此，我们总应扪心自问，是否已检查了计算结果。对于乘法，作者所讲的是小学所学的乘法检查方法。在德国将这种方法称为“Neunerprobe”，在英语中称为“**舍九法**(casting out nines)”，在法国称为 preuve par neuf。

将 a 的各位数字相加。如果和还不是一位数，则继续对各位数字相加，直到和是一个一位数(和是 9，要减去 9 得 0)，这个数就叫做 a 的校验和，我们用 s_a 表示校验和。下面给出一个示例：

$$4528 \rightarrow 19 \rightarrow 10 \rightarrow 1$$

对 b 和结果 c 进行同样的计算，可得到校验和 s_b 和 s_c。所有的校验和都是一位数。计算 $s_a \cdot s_b$，得到它的校验和 s，如果 s 不等于 s_c，则 c 不等于 a · b。在 AL-Khwarizmi 所著的代数书中对这个测试进行了描述。

让我们通过一个简单的示例来描述这个测试。设 a=429，b=357，c=154153。则 $s_a=6$，$s_b=6$，$s_c=1$。可得，$s_a \cdot s_b=36$，所以 s=9。因此，$s_c \neq s$，因此 s_c 不是 a 和 b 的积。实际上，正确积是 c=153 153，它的校验和是 9。因此，正确的积可以通过这个测试。但这个测试并非万无一失，因为 c=135 153 也能通过这个测试。虽然如此，这个测试已经相当有用了，它能检查出许多错误。

那么，这个测试所用的数学知识是什么呢？下面，我们介绍一种更通用的方法。设 q 是任意正整数；在上述方法中，q=9。设 s_a 是余数，在 a 除以 q 的整数除法中，即 $s_a=a-\lfloor a/q \rfloor \cdot q$，则 $0 \leqslant s_a < q$。在数学表示中，写作 $s_a = a \bmod q$[②]。同样，$s_b = b \bmod q$，$s_c = c \bmod q$。最后，$s=(s_a \cdot s_b) \bmod q$。如果 $c=a \cdot b$，则必然是 $s=s_c$ 这种情况。因此，$s \neq s_c$ 就证明了 $c \neq a \cdot b$，说明乘法做错了。如果 $s=s_c$，那意味着什么呢？我们知道，这意味着 q 能整除 c 与 a · b 的差。如果这个差为非零，则用不能整除差的任何 q，都能检查出乘法出错了。

让我们继续上述示例。取 q=7，则 $a \bmod 7=2$，因此，$s=(2 \cdot 0) \bmod 7=0$。但 $135153 \bmod 7=4$，则我们知道 $135153 \neq 429 \times 357$。

练习 1.4　解释，为什么作者在学校所学方法对应 q=9 的情况。提示：对于所有的 $k \geqslant 0$，$10^k \bmod 9=1$。

练习 1.5　(Elferprobe，舍 11 法)。10 的幂模 11，余数很简单，即对于所有的 $k \geqslant 0$，$10^k \bmod 11=(-1)^k$，即 $1 \bmod 11=1$，$10 \bmod 11=-1$，$100 \bmod 11=+1$，$1000 \bmod 11=-1$，以此类推。描述一种简单的测试，用于检查乘积模 11 的正确性。

---

① 原来奔腾芯片的浮点运算单元的除法算法的错误是臭名昭著的。这是由于算法所用的查找表中缺少少量表项所引发的。

② 根据定义 $sa=a-(\lceil a/q \rceil-1)q$，学校所学的方法使用的余数范围是 1 到 9，而不是 0 到 8。

1.4 递归版的学校方法

现在，我们推导学校方法的递归版本。这是我们遇到的第一个“**分而治之**”(**divide-and-conquer**)的范式，它是算法设计中的一个基本范式。

设 a 和 b 是两个 n 位整数，在乘法中作为被乘数和乘数。设 $k=\lfloor n/2 \rfloor$。将 a 分解成两个数 a_1 和 a_0，a_0 由 k 个最低有效位组成，a_1 由 $n-k$ 个最高有效位组成①。同样，也对 b 进行类似的分解。则，

$$a=a_1 \cdot B^k+a_0 \quad 且 \quad b=b_1 \cdot B^k+b_0$$

因此，

$$a \cdot b=a_1 \cdot b_1 \cdot B^{2k}+(a_1 \cdot b_0+a_0 \cdot b_1)B^k+a_0 \cdot b_0$$

公式给出了计算 $a \cdot b$ 的算法，具体步骤如下：

(a) 将 a 和 b 分解成 a_1、a_0、b_1 和 b_0。

(b) 计算 4 个积 $a_1 \cdot b_1$、$a_1 \cdot b_0$、$a_0 \cdot b_1$ 和 $a_0 b_0$。

(c) 将适当对齐的积相加得到 $a \cdot b$。

通过观察可知，数 a_1、a_0、b_1 和 b_0 都是$\lceil n/2 \rceil$位数字，因此，如果$\lceil n/2 \rceil<n$，即 $n>1$ 时，步骤(b)中的乘法比原来的乘法简单。现在，完成的算法如下。当乘一位数时，使用乘法原语。当乘 $n \geqslant 2$ 的 n 位数时，使用上述的三个步骤。

由上所述，能很清楚地知道为什么将这种方法称为分而治之。我们将 a 乘 b 的问题进行分解，形成一些更简单的同类问题。分而治之算法总是由三部分组成：第一部分，将原问题分解成更简单的同类子问题(步骤(a))；第二部分，使用相同的方法解决子问题(我们的步骤(b))；第三部分，从子问题的解决方案中，得到原问题的解决方案(我们的步骤(c))。

我们的递归整数乘法与学校方法有什么关系呢？答案是：它们确实是相同的方法。学校方法也计算如图 1.2 所示的 4 个部分积 $a_1 \cdot b_1$、$a_1 \cdot b_0$、$a_0 \cdot b_1$ 和 $a_0 b_0$。我们知道，递归整数乘法就是变相的学校方法，这意味着递归算法使用的原语操作次数也是平方级的。设我们也能从第一条原则中推导出这一结论。所以，我们能引入递推关系。递推关系是递归算法分析中非常重要的概念。

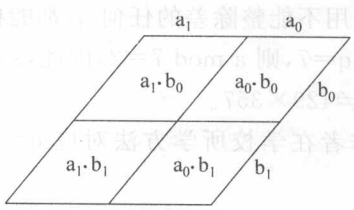


图 1.2 学校方法及其递归变量的示意图。菱形区域表示乘法 $a \cdot b$ 的部分积。这 4 个子区域对应着部分积 $a_1 \cdot b_1$、$a_1 \cdot b_0$、$a_0 \cdot b_1$ 和 $a_0 b_0$。在递归方案中，第 1 步，先对 4 个子区域的部分积求和，第 2 步，将 4 个所得和相加

① 通过观察可知，我们已经改变了表示方法：a_0 和 a_1 表示 a 的两部分，不仅代表位数了。