高等学校计算机科学与技术教材

计算机算法

□ 胡金初　主编
□ 汪存燕　副主编

- 原理与技术的完美结合
- 教学与科研的最新成果
- 语言精炼、实例丰富
- 可操作性强，实用性突出

清华大学出版社　●　北京交通大学出版社

高等学校计算机科学与技术教材

计算机算法

胡金初　主　编
汪存燕　副主编

清华大学出版社
北京交通大学出版社

·北京·

内 容 简 介

本书主要讲述、分析了各种算法的基本原理和解题技巧，以五种通用的算法设计技术为主线论述了分治策略、贪心策略、动态规划策略、分支限界法、回溯法等问题，对算法的时间和空间复杂性进行了分析。在内容的选材上注重基本理论和具体实例的结合，以便于读者理解。本书还对概率算法、近似算法、密码算法和 NP 问题进行了简单的介绍。

本书可作为计算机系本科学生及研究生的教材，也可作为计算机科学研究和软件开发技术人员的参考用书。

图书在版编目（CIP）数据

计算机算法/胡金初主编. —北京：清华大学出版社；北京交通大学出版社，2009.3
（高等学校计算机科学与技术教材）
ISBN 978 - 7 - 81123 - 560 - 9

Ⅰ. 计… Ⅱ. 胡… Ⅲ. 电子计算机-算法理论-高等学校-教材 Ⅳ. TP301.6

中国版本图书馆 CIP 数据核字（2009）第 035967 号

责任编辑：杨正泽
出版发行：清华大学出版社　邮编：100084　电话：010 - 62776969　http：//www.tup.com.cn
　　北京交通大学出版社　邮编：100044　电话：010 - 51686414　http：//press.bjtu.edu.cn
印 刷 者：北京瑞达方舟印务有限公司
经　　销：全国新华书店
开　　本：185×260　印张：13　字数：333 千字
版　　次：2009 年 3 月第 1 版　2009 年 3 月第 1 次印刷
书　　号：ISBN 978 - 7 - 81123 - 560 - 9/TP · 471
印　　数：1～4 000 册　定价：21.00 元

前　言

计算机算法是计算机学科的核心课程，熟练掌握这门课程后可以很好地解决在计算机研究中遇到的疑难问题，该课程的主要目的是培养读者分析和设计算法的能力，从而为编写高效的程序和开发优秀软件奠定基础。

本书主要介绍五种通用的算法设计技术，并且分析了各算法的基本原理和解题技巧。为了方便读者的理解，书中算法给出程序实现的方法和描述，读者可以用自己擅长的语言对这些算法进行上机实践以加深自己对这些算法的理解。全书共分16章，第1章介绍了算法设计和分析的基本概念，详细地对在一个表中搜索给定值和矩阵乘法两个算法进行算法分析，简单地介绍了算法的5种通用算法设计技术递归方程解的展开式；第2章介绍了几种常用的排序算法，从内部排序和外部排序两方面进行分析；第3章讨论了几种查找树的问题，包括二分查找树、2-3-4树、红黑树和B树；第4章讨论了图的算法，讲述了最短路径和最小生成树的问题；第5章介绍了串匹配问题，在该章中介绍了几种串匹配问题，如BM算法、KMP算法等；第6章介绍了五种算法设计技术之一的分治算法，解决二分搜索问题、最大最小元问题、大整数乘法问题、矩阵乘法问题；第7章介绍算法设计技术之二的贪心算法，解决背包问题、带时限的作业排序问题、单源最短路径问题、最小生成树问题、Dijkstra最短路径的优化算法；第8章介绍了算法设计技术之三的回溯法，解决n皇后问题、图的着色问题、0-1背包问题、哈密顿回路问题、子集和数问题；第9章介绍了算法设计技术之四的动态规划法应用在最长公共子序列问题、矩阵连乘问题等；第10章从0-1背包问题入手用分支限界法进行讲解；第11章简单地介绍了几种概率算法；第12章主要介绍了几何问题的实例；第13章介绍计算机算法的一个理论问题——NP问题，分析几个NP完全问题；第14章讲解了密码学算法，从背包公钥密码、RSA算法、数字签名等，对密码学作了简单的介绍；第15章介绍了近似算法；第16章重点介绍了一些数值问题的并行算法。在每章的结尾都有习题，以启发学生运用学到的知识解决实际问题。

从整体上看，本书具有内容全面、取材得当、实用和指导性强等特点，是作者多年来计算机算法课程教学的经验总结，是在授课讲义基础上，参考国内外有关材料编写而成。希望所有读者能从本书中体会到计算机算法的精髓所在，能对今后的工作和学习有所帮助。在教学内容的选择上也可以根据本校教学的实际情况节选部分内容。为了方便教师的教学，本书还配备有全套的教学幻灯片，可供教师在教学中选用。本书可作为高等院校的教科书或参考书，又可以作为计算机算法领域人员的参考书，还可以作为相关领域人员了解计算机算法知识的参考材料。

本书内容翔实、新颖、全面，书中所有提及的原理和概念都有相应的详细解释，并配有很多实例和插图帮助读者理解，以充实的内容在抽象概念和现实之间架设了桥梁，为读者深入理解计算机算法提供了理论基础。计算机算法是很有用的工具，同时也是一门比较难学的课程，可能令有些读者望而却步，本书用深入浅出的语言来讲解枯燥无味的概念，帮助读者

更好地理解课程的内容，通俗易懂的语言也便于读者自学。此书的写作遵照循序渐进原则，结合当今流行的计算机算法，配有许多实例和练习，逐步引导读者从一个门外汉变成一个既懂得计算机算法原理，又能够实际运用的熟练操作人员。

由于计算机算法这门课程所涉及的内容非常广泛，加上作者能力和水平所限，本书一定存在许多不足之处，恳请读者批评指正。

编　者

于上海师范大学

目　录

第1章

绪　论

计算机算法是对计算机上执行的计算过程的具体描述，是以一步一步的方式来详细描述计算机如何将输入转化为所要求的输出的过程。一个可以用算法解决的问题指的是对于任何输入，只要让这个编写好的计算机程序运行足够长的时间并给它足够多的空间，就能产生正确的答案。算法是一系列解决问题的清晰指令，也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。算法常常含有重复的步骤和一些比较或逻辑判断。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。

在计算机出现以前，数学家们就已经开始研究算法了，但此时的算法研究是作为数学的一个学科，是指为了解决一类实际或科学问题而概括出来的，带有一般性的计算方法，例如四则运算中的先乘除后加减，欧几里得的算法等，这一阶段的算法往往都以自然语言描述或数学公式的形式出现，我们称其为“原算法”。在计算机出现以后，此时的算法就不止是解决问题的过程和方法了，还需要用计算机能识别、执行的形式表达出来，可以称其为“计算机的算法”，现代意义上的“算法”通常是指可以用计算机来解决的某一问题的程序或步骤。

英国数学家阿兰·麦席森·图灵（Alan Mathison Turing）在一篇名为《可计算数学》的论文中首次提出了有关计算机的理论，其中提到的“存在任何计算机都无法解决的问题”，这个问题是指这个算法对于一个给定的输入在任何计算机上最终都无法停止。这个就是数学和计算机算法的区别，数学讲究的是精确性，而计算机算法讲究的是计算机上的可执行性，并不是所有的算法都可以用计算机在有限的时间里完成的，举例如下。

【例1-1】 给出26个英文字母，对这26个英文字母做全排列。这在数学上是一个简单的问题，但是用计算机来实现这一操作，是否可行呢？假设有一台高速计算机，能够每秒解决10^{13}个排列，共需要做26！个排列操作：

1. ABCDEFG……XYZ
2. BACDEFG……XYZ
3. BCADEFG……XYZ
4. BCDAEFG……XYZ

⋮

n. ZYX……DCBA

$n=26! \approx 4\times10^{26}$

1年$=3.153\ 6\times10^{7}$秒

$26!/(3.153\ 6\times10^{7}\times10^{13})\approx1.2\times10^{6}$（年）

由此可以看出如果用计算机算出全排列的结果需要用 1.2×10^6 年，也就是说我们不可能用这台计算机在有限的时间内解决这个问题，事实上，目前还无法用现在的任何一台计算机来解决这个问题。

算法是程序设计的精髓，程序设计的实质就是构造解决问题的算法，将其解释为计算机语言。算法是求解某个问题的有限指令序列，每条指令都是确定的、简单的、机械的、可执行的。对于任何一个属于这个问题的实例的有效输入，应在有限步（一步执行一条指令）内给出结果（输出），并中止。通俗地说，就是计算机解题的过程。

一个算法应该具有以下 5 个基本的特征。

① 输入：一个算法必须有零个或多个输入量。

② 输出：一个算法应有一个或多个输出量，把算法计算的结果输出。

③ 确定性：算法的描述必须无歧义，以保证算法的执行结果是确定的。

④ 有穷性：算法必须在有限步骤内实现。此处“有限”不同于数学概念的“有限”，天文数字般的有限（例如需要运行 100 万年时间）对于解决实际问题是无意义的。

⑤ 有效性：又称可行性。能够实现，算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

算法中常常含有重复的步骤和一些比较或逻辑判断的操作。如果一个算法有缺陷，或不适用于某个问题，执行这个算法就不能有效地解决这个问题。对同一个任务，在计算机上执行不同的算法，所需要的时间、空间及效率都会不同。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。所以一个算法的时间和空间复杂性就成为重要的问题，为了正确地描述一个算法的时间和空间复杂性，我们从数学的角度来讨论，而不考虑运行的计算机、语言、编程等。事实上，如果把算法的时间和空间复杂性问题的讨论放在某一台具体的计算机上（比如用 Intel 某一型号的 CPU），这样算法的运行时间就会受到具体的计算机性能的影响，这不利于把算法的理论问题说清楚。所以，对时间复杂性问题的讨论要归结为某些基本操作的次数问题，针对不同的问题可以选择不同的基本操作。基本操作的次数往往与问题的规模有关，在实际中确定问题的规模，一般考虑对基本操作的次数影响最大的量。

所以，算法必须满足正确性，即对于任意的一组输入，包括合理的输入与不合理的输入，总能得到预期的输出。如果一个算法只是对合理的输入才能得到预期的输出，而在异常情况下却无法预料输出的结果，那么它就不是正确的。

算法是由一系列具体步骤组成的，并且每一步都能够被计算机所理解和执行，而不是抽象和模糊的概念。每个步骤都有确定的执行顺序，即上一步在哪里，下一步是什么，都必须明确，无二义性。

无论算法有多么复杂，都必须在有限步之后结束并终止运行，即算法的步骤必须是有限的。在任何情况下，算法都不能陷入无限循环中。

1.1 算法的时间复杂性

算法的时间复杂性是指算法需要消耗的时间资源。一般来说，计算机算法是要解决问题

规模 n 的函数 $f(n)$，如果算法执行时间的增长率与 $f(n)$ 的增长率正相关，称为渐进时间复杂度（Asymptotic Time Complexity）。

算法是在有限步骤内求解某一问题所使用的一组定义明确的规则。通俗地说，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。一个算法在计算机上需要运行的时间往往是我们最关心的问题，而计算机运行的时间又在很大程度上与这一问题的输入规模有关，这里用算法的时间复杂性来讨论。

【定义 1-1】 如果一个问题的规模是 n，解这一问题的某一算法所需要的时间为 $T(n)$，它是 n 的某一函数，$T(n)$ 称为这一算法的“时间复杂性”。当输入量 n 逐渐加大时，时间复杂性的极限情形称为算法的“渐近时间复杂性”。

我们常用记号大“O”表示时间复杂性，注意它是某一个算法的时间复杂性。大 O 表示只是说有上界，如果 $f(n)=O(n)$，那显然 $f(n)=O(n^2)$ 也成立，它只给你一个上界，但并不是上确界，但人们在表示的时候一般都习惯表示前者。

此外，一个问题本身也有它固有的复杂性，如果某个算法的复杂性到达了这个问题复杂性的下界，那就称这样的算法是最佳算法。

“大 O 记法”：在这种描述中使用的基本参数是 n，即问题的规模，把复杂性或运行时间表达为 n 的函数。这里的“O”表示量级（order），是一种粗略的表示，并不精确。比如说“二分搜索的复杂性是 $O(\log_2 n)$”，也就是说它需要“通过 $\log_2 n$ 量级的步骤去搜索一个规模为 n 的数组”，为了简单起见用 $O(\log n)$ 表示，在本书中，如果不作特别说明，对数 $\log n$ 均指以 2 为底。记号 $O(f(n))$ 表示当 n 增大时，运行时间至多将以正比于 $f(n)$ 的速度增长。

这种渐进估计对算法的理论分析是非常有价值的，但它只能作大致的比较，在实践中使用时需要注意它的问题。例如，一个复杂性 $O(n^2)$ 的算法在输入规模 n 较小的情况下可能比一个 $O(n\log n)$ 算法运行得更快。但是随着 n 的增加，后一种具有较慢上升函数的算法必然工作得更快。下面来看几个分析时间复杂性的例子。

【例 1-2】

```
temp=i;
i=j;
j=temp;
```

解：以上三条单个语句的频度均为 1，该程序段的执行时间是一个与问题规模 n 无关的常数。算法的时间复杂度为常数阶，记作 $T(n)=O(1)$。如果算法的执行时间不随着问题规模 n 的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数。此类算法的时间复杂度是 $O(1)$。

【例 1-3】

```
sum=0;                    (1 次)
for (i=1;i<=n;i++)        (n 次)
    for (j=1;j<=n;j++)        (n² 次)
            sum++;            (n² 次)
```

解：$T(n)=2n^2+n+1=O(n^2)$。

【例 1-4】

```
for(i=1;i<n;i++)
{
y=y+1;          ①
for (j=0;j<=(2*n);j++)
    x++;        ②
}
```

解：语句①的频度是 $n-1$，语句②的频度是 $(n-1)\times(2n+1)=2n^2-n-1$，$f(n)=2n^2-n-1+(n-1)=2n^2-2$，该程序的时间复杂度 $T(n)=O(n^2)$。

【例 1-5】

```
a=0;
b=1;                    ①
for(i=1;i<=n;i++)       ②
{
  s=a+b;                ③
  b=a;                  ④
  a=s;                  ⑤
}
```

解：语句①的频度为 2，语句②的频度为 n，语句③的频度为 $n-1$，语句④的频度为 $n-1$，语句⑤的频度为 $n-1$，$T(n)=2+n+3(n-1)=4n-1=O(n)$。

【例 1-6】

```
i=1;          ①
while(i<=n)
  i=i*2;      ②
```

解：语句①的频度是 1，设语句②的频度是 $f(n)$，则：$2^{f(n)}\leqslant n$；$f(n)\leqslant\log_2 n$，取最大值 $f(n)=\log n$，$T(n)=O(\log n)$。

【例 1-7】

```
for(i=0;i<n;i++)
{
 for(j=0;j<i;j++)
     {
      for(k=0;k<j;k++)
        x=x+2;
     }
}
```

解：当 $i=m$，$j=k$ 的时候，内层循环的次数为 k；当 $i=m$ 时，j 可以取 0，1，…，$m-1$，所以这里最内循环共进行了 $0+1+\cdots+m-1=(m-1)m/2$ 次，所以，i 从 0 取到 n，

则循环共进行了 $0+(1-1)\times1/2+\cdots+(n-1)n/2=n(n+1)(n-1)/6$，所以，时间复杂度为 $O(n^3)$。

对于一个算法，我们应该区分它的最坏情况的时间复杂度和平均时间复杂度，它们之间是有差别的。如快速排序的最坏情况运行时间是 $O(n^2)$，但平均时间是 $O(n\log n)$。通过每次都仔细地选择基准值，就有可能把 n 平方情况［即 $O(n^2)$ 情况］的概率减小到几乎等于 0。在实际中，快速排序一般都能以 $O(n\log n)$ 运行时间来实现。下面是一些常用的记法。

访问数组中的元素是常数时间操作，即 $O(1)$ 操作。一个算法如果能在每个步骤操作后去掉一半数据元素，如二分搜索，通常它就能够在 $O(\log n)$ 时间内完成。用 strcmp 比较两个具有 n 个字符的串需要 $O(n)$ 时间 。常规的矩阵乘算法的时间复杂度是 $O(n^3)$，因为算出每个元素都需要将 n 对元素相乘并加到一起，所有元素的个数是 n^2。

具有指数时间的算法通常是需要求出所有可能结果。例如，n 个元素的集合共有 2^n 个子集，所以要求出所有子集的算法将是 $O(2^n)$ 的 。具有指数时间的算法一般说来太复杂了，除非 n 的值比较小。在现实世界中确实有许多问题（如著名的“售货员问题”），到目前为止找到的算法都是指数时间的。如果要在实际中解决这类问题，通常比较有效的方法是寻找近似最佳的算法替代之，近似算法将会在第 15 章中详细介绍。

1.2 算法的空间复杂性

算法的空间复杂度是指算法需要消耗的空间资源，其计算和表示方法与时间复杂度类似，一般都用复杂度的渐近性来表示。同时间复杂度相比，空间复杂度的分析要简单得多。与时间复杂度类似，空间复杂度是指算法在计算机内执行时所需存储空间的度量。记作：

$$S(n)=O(f(n))$$

一个算法的空间用量，依赖于这个算法的特定实现程序，一个程序要用存储空间来存放指令、输入数据、中间变量等，也要用一些工作空间来存储这个程序运行中的数据。输入数据可能有不同的结构，各种不同的数据类型需要不同量的存储空间。如果输入数据以一种自然形式存储，例如一个数组，我们考虑除了程序和输入数据需要的存储空间以外，还需要用到一些额外的空间量来存放中间结果，若这些额外需要的空间量是常数，与问题的输入规模无关，就称该算法为就地工作。有些问题的输入数据能用不同形式的数据结构表示，例如图在计算机中可以有邻接矩阵、邻接表等几种不同的表示方法，在考虑输入数据本身所要的空间时同时还考虑算法需要的额外空间，一般是指正常占用内存的开销，以及辅助存储单元的大小。讨论方法与时间复杂度类似，不再赘述。

1.3 两个算法的分析实例

一个算法的优劣是用空间复杂度与时间复杂度来衡量的。对时间和空间复杂性的讨

论，应该进行抽象化，脱离具体的计算机机型、采用的编程语言及算法描述的语句长度等。这样可以将算法的时间复杂性问题的讨论聚焦到采用的方法上，为此引入基本操作的概念，然后通过比较基本操作的次数，来判定一个算法的优劣。算法的基本操作次数往往与问题的规模有关，那么如何首先确定一个问题的规模？一般选择对基本操作的次数影响最大的量。

两个算法的基本操作和规模如表 1-1 所示。

表 1-1 两个算法的基本操作和规模

问 题	基本操作	规 模
在一个表中搜索给定的 x	比较两个数的大小	表的长度
矩阵乘法	两个数相乘	矩阵的阶

【例 1-8】 搜索有序表的分析。

L 是一个有 n 个元素且以非减次序存放的表，x 是给定要搜索的元素。

输出：整数 j，使得当 x 在 L 表中时，$L[j]=x$，如果 x 不在表中，则 $j=0$。

```
j=1;
while(j<n)and(L[j]<x)
    {j++;}
if(L[j]<>x)
  then j=0;
```

算法分析：显然这个算法的最坏情况只有当 x 不存在于 L 表中，且 x 大于 L 表中的最大值时达到，最坏情况 $W(n)=n$，这个问题的输入为 L，x 在 L 表中的位置有 $(n+1)$ 种可能，令 I_i 表示 x 在 L 中第 i 个位置的输入，I_{n+i} 表示 x 不在 L 中的输入，也可以分为 $(n+1)$ 种情况：$x<L(1)$，$L(1)<x<L(2)$，…，$L(i-1)<x<L(i)$，…，$L(n)<x$ 共 $(n+1)$ 个间隙，记落在这些间隙的情况为 I_{n+i}，则当 x 在表中时，比较时间为 $t(I_i)=i$，当 x 不在表中时，落在这些区间的比较时间为 $t(I_{n+i})=i$，$(1\leqslant i\leqslant n)$，$t(I_{n+i})=t(I_{n+n+1})=t(I_{2n+1})$，$i=n+1$，设 x 在 L 中的概率为 q，则不在表中的概率为 $1-q$，而且假设 x 在 L 中每个位置出现的概率相同，x 落在这些间隙的概率也相同，则 $p(I_i)=q/n$，$p(I_{n+i})=(1-q)/(n+1)(1\leqslant i\leqslant n+1)$，则平均时间复杂度的式子为

$$\begin{aligned}A(n)&=\sum_{i=1}^{n}(q*i/n)+\sum_{i=1}^{n}[(1-q)*i/(n+1)]+(1-q)*n/n+1\\&=n/2+1-q/2-(1-q)/(n+1)\approx n/2\end{aligned}$$

由上述公式可以看出，算法的平均搜索次数总是约等于 $n/2$，而与 q 无关。

【例 1-9】 矩阵相乘。

矩阵 $\boldsymbol{A}$ 乘以矩阵 $\boldsymbol{B}$，得到矩阵 $\boldsymbol{C}$，写作 $\boldsymbol{C}=\boldsymbol{AB}$。

输出是矩阵 $\boldsymbol{C}$，算法如下：

```
for(i=1,i<=n,i++)
  {
    for(j=1,j<=n,j++)
```

```
    {
      c[i,j]=0;
      for(k=1,k<=n,k++)
       c[i,j]+=a[i,k]*b[k,j]
    }
}
```

算法分析：每个元 $c[i, j]$需要做 n 次乘法，共有 n^2 个元，所以 $A(n)=O(n^3)$，因此可以得出，该算法的时间复杂性与数据本身无关。

1.4 算法设计技术

算法设计的基本要求是：首先是正确性（Correctness），程序中不含语法错误，程序对一切合理的输入数据都能产生满足要求的结果，程序也能对不合理的输入数据产生符合规格要求的结果。其次，算法要具有可读性（Readability），因为研究算法的目的是为了阅读和交流，可读性有助于对算法的理解，可读性有助于对算法的调试和修改。算法应具有高效率与低存储量，处理速度要快，存储容量要小，有时候处理时间和存储空间是矛盾的，实际中，往往是求得时间和空间的折中。

多年来，人们提出了一些通用的算法设计技术如分治方法、贪心法、回溯法、动态规划法、分支限界法等，我们用它们解决了许多类的问题，产生有效的算法，其中贪心法、动态规划法和分支限界法大多用来解决最优化的问题。本小节简单地来介绍一下这五种设计方法，具体算法将会在第 6、7、8、9、10 章分别介绍。

1.4.1 分治方法

分治法的设计思想是将一个难以直接解决的大问题分割成一些规模较小但类型相同的问题，以便各个击破，分而治之。分治方法适用于许多领域，尤其在非数值计算领域中应用得十分广泛。所谓分治法，是指对于一个输入规模为 n 的函数或问题，用某种方法把输入分割成等价的 k 个规模分别为 n/k 的子问题。首先解出 k 个子问题，然后再将这些子问题的解用某种方法组合起来形成原问题的解。如果划分后的子问题仍然很大，那么再对这些子问题反复使用分治法，直到最后的子问题分得足够小，不必再进行分割，就可以很容易得出它们的解。由于在使用分治法时，仅仅是在规模上较小，产生的子问题的类型往往和原问题的类型相同，因而通常可以采用递归过程来进行算法设计。

任何一个可以用计算机求解的问题所需的计算时间都与求解问题的规模 N 有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如，对于 n 个元素的排序问题，当 $n=1$ 时，不需任何计算；$n=2$ 时，只要作一次比较即可排好序；$n=3$ 时，只要作 3 次比较即可……而当 n 较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时是相当困难的。

如果原问题可分割成 k 个子问题（$1<k\leqslant n$），且这些子问题都可解，并可进一步利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解的程度。这自然导致递归过程的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。分治法所能解决的问题一般具有以下几个特征。

① 该问题的规模缩小到一定的程度就可以容易地解决；

② 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

③ 利用该问题分解出的子问题的解可以合并为该问题的解；

④ 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增大而增加；第二条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用；第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑贪心法或动态规划法；第四条特征涉及分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但一般用动态规划法较好。分治法在每一层递归上都有 3 个步骤。

① 分解：将原问题分解为若干个规模较小、相互独立、与原问题形式相同的子问题。

② 处理：若子问题规模较小而容易解决则直接解，否则递归地解各个子问题。

③ 合并：将各个子问题的解合并为原问题的解。

具有递归过程的分治法有非常重要的价值，许多问题的求解都可以采用递归过程的分治法，例如二分查找、大整数相乘、矩阵乘法、快速排序等问题的求解算法，都需要应用分治法。本书将会在第 6 章对大整数乘法和矩阵乘法如何运用分治法求解作详细介绍。

分治策略就是“分而治之”的意思。分治策略一般可以递归进行，作为算法设计的技巧来说，递归是著名的“分治”策略。分治策略的一个典型应用是如下的递归结构。

已知问题 P 可解。若 P 能分为较小的子问题，则：

1. 将 P 划分为多个等价的部分：P_1，P_2，…，P_k；

2. 解 P_1；

3. 解 P_2；

⋮

k. 解 P_k；

$k+1$. 将 k 个部分的解组合成原问题 P 的解。

否则，直接解 P。

为了分析递归算法的时间或空间复杂性，通常把描述整个问题的时间或空间复杂性函数的表达式写成较小问题样本的时间或空间复杂性函数。

【例 1-10】 设 X 和 Y 是两个 n 位的二进制数，按照传统乘法需要 $O(n^2)$ 次“位”运算，使用分治法可以降低到 $O(n^{1.59})$。以 n 是 2 的幂为例，把 X 和 Y 分为两半，如图 1-1 所示。

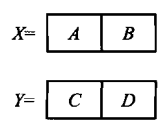

图 1-1 X 和 Y 分半