普通高等教育“九五”国家教委重点教材

数据结构

——C++与面向对象的途径

（修订版）

张乃孝　裘宗燕

高等教育出版社

图书在版编目(CIP)数据

数据结构:C++与面向对象的途径/张乃孝,裘宗燕
—修订版,-北京:高等教育出版社,2001
ISBN 7-04-009203-4

Ⅰ.数… Ⅱ.①张…②裘… Ⅲ.数据结构,面向对象的
Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(2000)第 77237 号

数据结构 C++与面向对象的途径(修订版)
张乃孝 裘宗燕

出版发行	高等教育出版社		
社 址	北京市东城区沙滩后街 55 号	邮政编码	100009
电 话	010-64054588	传 真	010-64014048
网 址	http://www.hep.edu.cn http://www.hep.com.cn		
经 销	新华书店北京发行所		
印 刷	北京印刷集团有限责任公司印刷二厂		
开 本	787×1092 1/16	版 次	1998 年 6 月第 1 版 2001 年 1 月第 2 版
印 张	25.25	印 次	2001 年 1 月第 1 次印刷
字 数	600 000	定 价	21.30 元

本书如有缺页、倒页、脱页等质量问题,请到所购图书销售部门联系调换。

再版前言

本书是一本系统介绍数据结构的教材。在本书的论述过程中会反复联系到计算机领域的许多重要问题，包括问题求解、算法的分析与设计、抽象数据类型、程序设计方法、程序设计语言等内容。根据目前计算机科学发展的情况，本书采用面向对象的思想组织与课程有关的内容，运用 C++语言作为数据结构和算法的描述语言，其目的是为了保证本书的先进性和适用性。

数据结构的一般讨论并不依赖于具体的语言，然而，在具体问题求解时，使用什么语言描述数据结构会在很大程度上影响程序设计者的思维方式和方法。选择一种实用的程序语言作为工作语言，有利于使讨论更面向实际应用，使参加课程学习的学生不仅可以掌握数据结构的抽象概念和性质，也有助于了解实现的思想和方法。

由于程序设计语言的发展，适合用来讨论数据结构的语言越来越多，今天国际上新的数据结构教材已经不再用伪语言作为工具了，随着面向对象方法的广泛应用，许多学校已经把 C++语言定为学习计算机的第一种程序设计语言，越来越多的教材采用 C++作为数据结构的教学语言。

在本书中，不仅讨论了抽象的数据结构，而且讨论了各种数据结构的不同实现方式并对它们进行了比较，从中可以看到不同数据结构的相互关系，包括结构上的（逻辑）关系和实现上的（表示）关系。采用 C++语言和面向对象的方法讲解数据结构，对这些关系的讨论提供了有力的工具。

由于 C++语言的一些优点，特别是它的模板机制，使我们可以比较清晰地讨论具有类型参数的数据结构。例如，我们实现的不是整数的堆栈，也不是字符串的堆栈，而是一般的具有类型参数的堆栈。一个实现可以满足各种程序对于堆栈的需要。这样学习的数据结构不仅更接近那种抽象概念上的数据结构，同时也可以直接用到不同类型对象的相同计算过程中去。

本书的讨论力图反映计算机科学的最新发展，用面向对象的方法组织数据结构的主要内容，以抽象数据类型的方式定义各种结构的用户界面，以时间和空间开销作为评价各种结构使用好坏的主要标准，从简到繁，系统地介绍各种常用的数据结构，强调不同结构之间的联系。书中采用 C++语言描述各种类的界面和实现。把面向对象的程序设计方法与数据结构的主要原理紧密结合起来，讲解过程中还插入大量实例。读者通过本书的学习不仅能掌握数据结构的基本原理和基本方法，同时可以把所学的知识用于实际问题求解的过程之中。

本书假定读者有基本的计算机基础和使用知识，具有基本的程序设计经验，掌握了 C 语言程序设计的基本技能（例如，学过一个学期 C 语言）。本书适合作为大学本科计算机专业或相关专业数据结构课程的教材，或作为其他专业的计算机提高课程教材，也可以作

为大学本科或专科有关专业面向对象程序设计课程的教材，或程序设计实践课程的教材和参考书。对于超出教学大纲的或难度较大的章节，书中已用星号（*）标出，供教学时参考。

全书共分 12 章:

第一章，绪论。简单讨论了数据结构和算法在计算机科学中的地位和作用，以及它们与程序设计方法，程序设计语言，抽象数据类型等相关领域的联系。

第二章，C++与面向对象初步。主要介绍面向对象程序设计（OOP）的一些重要概念和C++语言的基本特征。通过本章的学习，读者能了解：什么是抽象数据类型；什么是类；什么是对象；C++中的类与抽象数据类型的关系；面向对象的程序设计等。

第三章，字符串——数据封装技术。本章定义了一种更安全可靠的字符串类型，克服了 C 语言的不足。同时也以字符串作例子，讨论数据抽象和封装的有关问题，介绍 C++语言中这方面的特征，以及这些特征的使用方法。

第四章，向量——程序重用技术。本章建立了一种安全可靠的向量数据类型，并为它实现了许多有用的操作；为了构造对复杂结构类型的循环，引入抽象遍历器类，并用继承方法定义了向量遍历器类；最后给出了几个主要的向量排序算法。通过这一章的学习，读者应该掌握通用程序模块的实现技术和不同途径的重用技术。

第五章，动态数据结构——链表。主要讨论了各种常用的链表结构及其实现方法。链表和向量是最基本的数据结构，是后面许多复杂数据结构的实现基础。

第六章，栈和队列。介绍了栈和队列的抽象概念、具体实现及其应用。

第七章，树和二叉树。介绍了树和二叉树的概念，重点介绍二叉树的实现，及树结构用于快速检索的一些技术。

第八章，优先队列。优先队列用于支持快速查找、存取和删除集合中最小元素，它可以用表、树、向量等结构实现。本章主要介绍了堆和斜堆的概念，以及通过它们实现优先队列的方法。

第九章，集合和字典。本章介绍了集合和字典的概念，以及实现它们的各种实用技术。

第十章，散列表。本章介绍散列表及其实现。特别讨论了不同的关键码具有相同的散列值（碰撞）的处理问题。

第十一章，图。这一章里讨论图的概念，并且考虑图的几种不同实现方式，介绍了图上的一些重要算法和应用。

第十二章，文件。介绍文件的概念，包括流文件及其操作，把文件作为数据结构实现机制的方法，用文件实现字典的一些问题，特别是索引结构等。

感谢

本书的形成包含着许多人的思考和贡献。作者曾用本书的初稿在北京大学校内讲授过数据结构课程，数学学院九五和九六级的同学和数学学院参加课程辅导的研究生们就这个课程和本书的初稿提出了许多宝贵意见。在本书的修改和整理过程中刘海燕同学做了大量输入和整理工作。另外，书中附录 C 的“多叉路口的交通管理系统”上机报告由俞欢同学完成，经陈利立整理。作者特别感谢北京航空航天大学麦中凡教授和北京大学韩玉真教授，

他们认真地审阅了全部书稿，并提出了许多中肯的意见。在与高等教育出版社的共同努力下，本书初版于 1998 年上半年与读者见面。两年来，在使用过程中，著者听到许多读者热情的赞扬和鼓励，也得到许多非常宝贵的意见和十分具体的建议。为了感谢广大读者的支持，我们总结了近年来的教学经验，及时对本书的内容做了修订。

修订工作主要涉及以下几个方面：

- 把第一版前言的主要内容改写为新版的第一章；
- 把原来的三、四两章调整合并为新版的第四章；
- 改写原来的九、十两章，调整了一些内容的前后次序；
- 适当补充了各章的例子、习题，并增加了若干上机实习题；
- 增加了一个上机实习报告的例子，作为附录；
- 订正了原来文字和程序中的一些错误。

由于面向对象是一个比较新的领域，在如何用面向对象的观点讲授数据结构方面作者的经验仍然不足，需要进一步思考和探索。因此在本书中难免会还有一些错误和不足之处，恳请读者批评指正。

张乃孝　裘宗燕

2000 年 4 月于北京大学

目　录

第一章 绪 论

数据结构讨论的是计算机科学技术领域里一些基本的问题，它是计算机教育专业中的一门核心课程，也是一门理论和实际紧密结合的基础课。随着计算机科学技术的飞速发展，计算机的许多领域都发生了很大的变化，数据结构的教学内容和教学方法也随着不断更新。但是，数据结构课程在计算机科学教育中的重要地位和作用并没有改变。

数据结构也称“信息结构”。对于复杂一点的实际问题，往往需要处理大量数据，这些数据之间经常又有着错综复杂的联系。在计算机中怎样才能有效地表示（存储）这些数据及其相互联系，使之能够在相关的处理程序中有效地用来模拟和解决问题，这就是数据结构课程讨论的中心问题。著名计算机科学家 P. Wegner 认为：计算机科学就是“一种关于信息结构转换的科学”。因此学好数据结构对于深入理解计算机科学，继续学习计算机的其他课程是非常重要的。无论是从事计算机实际开发工作，还是理论研究工作，本课程都是必不可少的基础。

数据结构与计算机科学的许多领域都有密切的关系，在学习过程中，了解它与本学科其他课程的联系，明确其在学科领域中的位置，有助于抓住学习的要领，理解课程内容的实质。下图形象地显示了数据结构在用计算机解决问题过程中，与计算机科学技术其他相关领域的关系。

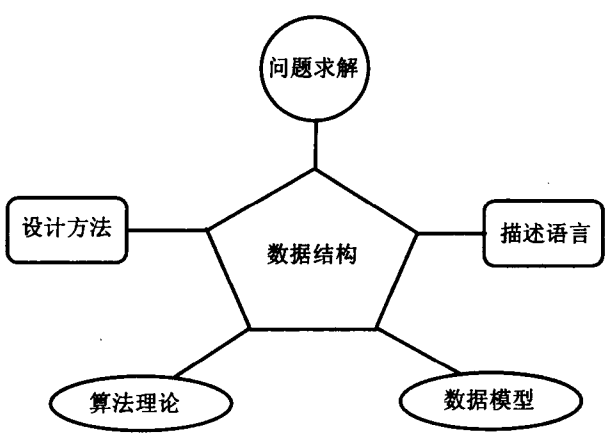

图 1.1 数据结构的地位及其与相关领域的联系

1.1 问 题 求 解

用计算机解决实际问题，就是在计算机中建立一个模拟解决问题的模型。在这个模型

中，计算机内部的数据表示了需要被处理的实际对象，包括其内在的性质和关系；处理这些数据的程序则模拟对象领域中的求解过程；通过解释计算机程序的运行结果，便得到了实际问题的解。

1.1.1　问题

首先看一个实际问题的例子。

考虑一个多叉路口（见图 1.2）的问题。在这个路口中，共有 5 条道路相交，其中 C 和 E 是单行线，其他为双行线。为了设计一个交通信号灯的管理系统，首先需要分析一下所有车辆的行驶路线的冲突问题。这个问题可以归结为对车辆的可能行驶方向作某种分组，对分组的要求是使任一个组中各个方向行驶的车辆可以同时安全行驶而不发生碰撞。显然对这个路口存在许多不同分组方案，而如果分组越少，可以同时行驶的车辆也就越多，从而使管理系统的质量越高。

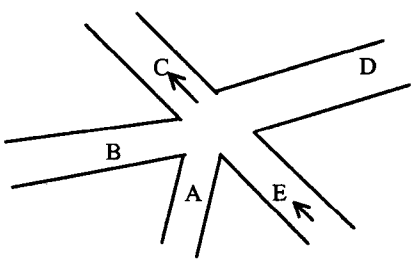

图 1.2　一个交叉路口的模型

1.1.2　问题的分析

根据这个路口的实际情况可以确定 13 个可能通行方向：A→B，A→C，A→D，B→A，B→C，B→D，D→A，D→B，D→C，E→A，E→B，E→C，E→D。有些方向明显不能同时进行，如 A→B 与 B→C 等。为了叙述清楚和方便，把 A→B 简写成 AB，用一个结点表示（一个小椭圆），在不能同时行驶的结点间画一条连线（表示它们互相冲突），便可以得到图 1.3 所示的图式。这样得到的表示可以称之为“图”（图是数学分支“图论”研究的对象，也是一种典型的数据结构。在本书的第十一章将讨论这种结构）。

上面这样做的结果是把一个实际问题变成了另一个问题，把要解决的问题借助图的模型清楚而严格地表达出来。这个问题就是：要求将图 1.3 中的结点分组，使有线相连（互相冲突）的结点不在同一个组里。进一步还希望对于这个问题能够设计一个最佳（分组数最少）的方案。习惯上把这种解称作“最优解”。

由于上面分析过程用了图的抽象，使得表达出的问题已经比最初提出的交通管理问题更一般、更抽象了，它反映了更大一类问题的要求。例如，如果把上图中的一个结点理解为一个国家，结点之间的连线看作两国有共同边界，上述问题就变成著名的“着色问题”：即求出最少要几种颜色可将图中所有国家着色，使得任意两个相邻的国家颜色都不相同。

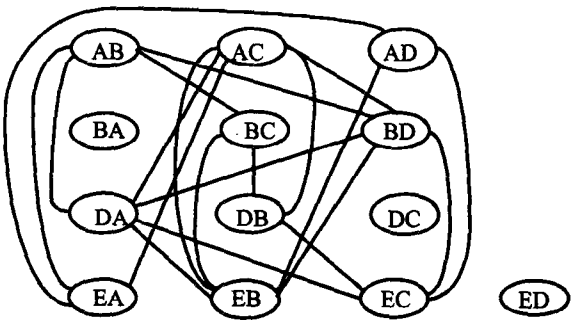

图 1.3 图 1.2 交叉路口的图式模型

1.1.3 算法的选择

对于一般的着色问题要求最优解，可选的一个解法是：从 1 开始逐个穷举出所有可能的组合，检查这样的分组是否满足要求，首先满足要求的分组，自然是最优解。

假设求解的图中有 n 个结点，首先考察如果放在一个组里（显然只有一种组合方式）行不行？即考察组里的结点是否有线相连？如果没有，最优解已经找到；否则考察如果放在两个组里行不行？注意逐个穷举出所有可能的分成两个组组合数，可能很大：例如两个组按 1:n–1 分配，就有 $C_n^1=n$ 种，按 2:n–2 分配，就有 $C_n^2=n(n-1)$种，等等。当考察了所有放在两个组里可能的组合后都不行的话，接着考虑分为 3 组、4 组的各种组合。

由此可见，这类穷举法对结点少的问题（称为“规模小的”问题）还可以用；对规模大的问题，由于求解时间会随实际问题规模的增长而指数性上升，使计算机无法承受。因此人们对这类问题一般采用求近似最优解的方法处理。

求着色问题的近似解，一种常用的方法称为“贪心法”，思想为：先用一种颜色给尽可能多的结点上色；然后用另一种颜色在未着色结点中给尽可能多的结点上色；如此反复直到所有结点都着色为止。

选用一种新颜色给结点上色时要做以下工作：

1. 选出一个未着色的结点并用该新的颜色上色。

2. 寻找那些仍未着色的结点，如果某结点与用新颜色着色的结点没有边相连，则可将这个结点用该颜色上色。

把这种方法应用到关于交叉路口的例子中，可能得到如下一种分组（实际上，根据处理的顺序不同，可能得到不同的分组）：

（1）红色：AB AC AD BA DC ED

（2）蓝色：BC BD EA

（3）绿色：DA DB

（4）白色：EB EC

这个解说明，如果按这种方式对车辆行驶方向进行分组，就可以保证不会发生车辆相

撞的事件。

1.1.4 解的精化

现在的问题是如何将求解着色问题的这种想法在计算机上实现，也就是要写出一个程序，对于任一个交叉路口的管理问题（实际上，只要一个问题可以化为着色问题）就可以用这个程序求解。

为此需要做以下工作：首先，为问题中所有有关数据设计适当的表示形式，不仅包括需要表示的结点和连接，可能还有为计算过程的实现而用的辅助性数据结构。然后选择一种适当的程序设计语言实现这些数据结构，并在设计好的数据结构上精确地描述上面提出的算法，完成一个程序，使之能在计算机上运行。

对于这个具体问题，如果有一种很高级的语言，它可以直接描述和处理抽象的集合和图，那么程序的实现就非常简单。假设需要着色的图是 G，集合 V_1 包括图中所有未被着色的结点，着色开始时 V_1 是 G 所有结点集合。NEW 表示已确定可以用新颜色着色的结点集合。从 G 中找出可用新颜色着色的结点集的工作可以用下面的程序框架描述：

```
置 NEW 为空集合;
for 每个 v ∈ V1 do
    if  v 与 NEW 中所有结点间都没有边
        从 V1 中去掉 v;  将 v 加入 NEW;
```

本程序片段每执行一遍，集合 NEW 中就得到可以用一新颜色着色的一组结点。着色程序的实现就是反复执行这个程序段，直到 V_1 为空。每次执行选择一种新颜色，程序段执行的次数就是需要的不同颜色个数。这个程序片段里涉及对集合和图的操作，包括由集合中去掉一个元素，向集合里增加一个元素，检查两个结点之间在图中是否有边连接等。有了这些结构和操作，程序的实现非常简单。

但是，常见程序设计语言并不直接支持图和集合等数据结构，通常只提供一些基本数据类型（例如整数、实数、字符等）和一些数据构造手段（例如数组、记录、指针等）。要解决这个计算问题，必须自己用选定的语言提供的机制实现这些结构（集合、图等）和操作（对集合的元素增删、对图的边的判断等），这些正是本书将要讨论的基本内容。

实际上，对这个问题的讨论具有普遍性。在用计算机对问题求解过程中，人们总是首先分析问题的需求，抽出抽象模型，然后设计适当的数据结构和有关的算法，最后用一种程序语言精确描述所需要的数据和算法，实现所需要的程序。

1.2 数 据 结 构

数据结构的研究虽然只有短短几十年的历史，但已经取得十分丰富的成果。为了学习

和研究的方便，计算机科学家把常用的数据按照它们的表示方法进行分类，并对各种结构的行为特性做了深入研究，总结出许多典型的数据结构。本书的主要内容就是讨论这些典型的数据结构。

在对各种数据结构进行分类讨论的时候，重点关心的是下面3个方面：

第一，这个结构中各元素之间的逻辑（或抽象）关系，在有的教材中简称为“逻辑结构”。例如，向量中各元素之间是一种简单的“线性”关系，所以称它的逻辑结构是线性结构，或简称向量是一种线性结构。类似地还有“树形结构”、“复杂结构”等。

第二，这个结构中各元素之间的存储方式，在有的教材中简称为“存储结构”。例如，向量中各元素如果是按其逻辑关系“顺序”存储，就称为是“顺序表示”或“顺序结构”；如果向量中各元素是通过指针链接存放在内存，就称为“链接结构（表示）”。类似还有“索引结构（表示）”、“散列结构（表示）”等。

第三，这个结构的行为或特性。这主要体现在与这个结构相关的运算集合的定义上。在数据结构中，有些结构就是根据它们的不同行为加以不同的命名，例如“栈”和“队列”的主要区别在于执行插入和删除运算的定义不同。另外，通常称为的“动态结构”也是指一大类可以方便地进行插入删除操作的数据结构，如链表；而对应的所谓“静态结构”是指一大类不适合频繁进行插入删除操作的结构，如（顺序表示的）向量。

需要特别指出的是，由于一种逻辑结构通常可以采取多种存储表示，加上计算机的概念缺少严格的规范定义，数据结构中的一个名词在不同的地方可能会表示不同的含义。例如“向量”一词，在某处可能只是强调其抽象逻辑结构；在另外一个地方可能是指按某种方式存储的结构（例如“顺序方式表示的向量”）；甚至在别的地方是指一个具体的对象（例如是一个“顺序方式表示的向量类型的变量”）。因此读者需要根据上下文来确定它们的含义。

下面，对本书中将要讨论的各种主要结构做一简单介绍：

“向量”是一种线性结构，向量的每个元素排定一个编号，称为“下标”。本书中的向量均指采用顺序表示的向量，它的一个显著的特点是所有元素都可以通过下标直接访问。向量适合表示那种元素个数已知的或者能够基本确定其数量范围的对象。

“字符串”也是一种线性结构，它以字符为元素。本书中的字符串均指采用顺序表示的形式，因此每个字符可以直接访问。字符串的特点是每个成员只占很少的存储空间，因此，如何有效地实现字符串是研究的重点。

“链表”（简称为“表”）是一种动态数据结构。链表中的元素通常需要频繁地插入和删除，适合于表示事先无法确定数目的元素序列。对表元素的使用可以从表的一端开始顺序地逐个进行。表的第一个元素称为“表头”，最后一个元素称为“表尾”。虽然表中的每个元素在表里有一个顺序排列位置，但对它们不能由位置进行直接访问。

“堆栈”和“队列”是两种常用的数据结构，可以向其中存入元素，或从中取出元素。堆栈元素的存入和取出按照后进先出原则，最先取出的总是在此之前最后放进去的那个元

素。队列实现先进先出的原则，最先到达的元素也最先离开队列。堆栈和队列都是非常重要的数据结构，在计算机领域的各个方面应用非常广泛。

“树”和“二叉树”结构统称“树形结构”，逻辑上表示了结点的层次关系，比向量复杂。树最上面一层只有一个元素，称为“树根”。每个元素可以有若干相关联的下层元素，这些元素被称为是该上层元素的“子结点”；每个下层元素至多有一个对应的上层元素，称为它的“父结点”。许多实际的和理论的问题中都可以抽象出某种树形结构来。

“集合”是一个数学概念，也是一种基本数据结构。集合数据结构的特点在于许多运算是对整个数据结构进行的，其基本运算包括求并、求交、求差集、求子集、和相等性检测等。集合是（与顺序无关的）元素的汇集。

“散列表”可以看作一种广义的向量，元素的访问同样通过下标进行。但是，散列表的下标可以是非整数类型，通过一个函数（称为“散列函数”）把下标值映射到表示存储地址的整数值，然后再进行直接存取。散列表的主要特点是可以实现对大量离散元素的有效存储和快速访问。

“字典”（又称为“映射”）可以看作是一种关联的集合，每个关联包含着一个下标（或称关键码）和一个值。所以抽象地看，一个字典就是由关键码集合到值集合的一个映射。字典也可以看为是一种具有下标性质的结构。它的下标也允许采用任何类型的值。字典在计算机领域应用广泛，人们提出了这种结构的许多实现方法。

“图”是一种较复杂的结构，它包括一个结点集合和一个边集合，边集合中每条边联系着两个结点。信息可以存储在结点里，也可以存储在边里。许多实际问题中的数据可以用图表示，如公路网络、通信网络、不同事物间的联系，等等。

1.3 算　法

算法是为实现某个计算过程而规定的基本动作的执行序列。一个算法可以有 0 个或者多个输入，这些输入数据是在算法开始时提供的一组量。对算法的描述应该精确地说明这些输入的个数、类型以及它们应满足的初始条件。算法的每个步骤必须明确描述，并且可行，不能有二义性。

算法正确性的含义是指它能够完成意想中的操作。关于正确性通常的提法是：如果一个算法以一组满足初始条件的输入开始执行，那么该算法的执行一定终止，并且能够得到满足要求的结果。

算法是一个十分古老的研究课题，然而计算机的出现为这个课题恢复了青春和活力，使算法的设计和分析成为计算机科学中最为活跃的研究热点之一。

1.3.1 算法的设计

在实际应用中，算法的表现形式千变万化，但是算法的情况也和数据结构类似，许多算法的设计思想具有相似之处，可以对它们分类进行学习和研究。例如，本章第一节提到

的一种算法称为“贪心法”，其基本思想是：当追求的目标是一个问题的最优解时，设法把对整个问题的求解工作分步骤完成，在其中的每一个阶段都选择那种从局部看是最优的方案，以期通过各阶段的局部最优选择达到整体的最优。当然，贪心法实际上不能保证都成功地产生一个全局性最优解，但是通常可以得到一个可行的较优解。

另一种常用的算法设计方法称为“分治法”，其基本思想是：把一个规模较大的问题分成两个或多个较小的与原问题相似的子问题。首先对子问题进行求解，然后设法把子问题的解合并起来，得出整个问题的解，即对问题分而治之。如果一个子问题的规模仍然比较大，不能很容易地求得解，就可以对这个子问题重复地应用分治策略。

“二分法检索”就是用分治策略的典型例子。如果要在一个整数的有序数组（可以假设元素按增序排列）中找一个数，要求查出这个数在这个数组里的位置，二分法检索采用的方法是先找到数组中居于中间位置的元素，将该元素与所找数字进行比较，如果所查数字比较小，那么它一定不会出现在中间元素的右边，下面只需在左半个数组中检索。反过来，若所查数字较大，则不必在左半个数组中继续寻找，只需在右半个数组中检索。这样，通过一次比较就能够排除掉一半元素。再做下去，只要对剩下的半个数组重复使用同样方式，又可以再排除掉四分之一的元素，如此重复若干次就能很快确定整数的存在与否，并确定如果存在时它在什么位置。

还有一类常用算法被称为“回溯法”。有一些问题，只能通过彻底搜索所有可能情况寻找一个满足某些预定条件的最优解。由于彻底搜索的运算量通常非常大，往往大到使用计算机也不能在合理时间内得到解。回溯算法是处理这类问题的一个方法，基本思想是一步一步向前试探，当有多种选择时可以任意选择一种，只要可行（或暂时没有失败）就继续向前，一旦发现问题或失败就后退，回到上一步重新选择，借助于回溯技巧和中间判断，常常可以大大地减少搜索时间。常见的迷宫问题以及八皇后问题都可以用回溯方法来解决。

除上面提到的几种典型方法外，常用的算法设计方法还有“动态规划”、“局部搜索”和“分支限界”等。

本书在讨论数据结构的同时，自然涉及与数据结构相关的许多算法，掌握这些算法的设计方法和共性，对于提高程序设计的水平是十分重要的。

1.3.2 算法的分析

评价一个算法优劣的重要依据是看这个算法的执行需要占用多少机器资源。在各种机器资源中，时间和空间是两个最主要的方面。因此，在进行算法分析时人们最关心的就是算法的运行所要付出的时间代价和空间代价。

算法的空间代价（或称空间复杂性）：当被解决问题的规模（以某种单位计算）由 1 增至 n 时，解该问题的算法所需占用的空间也以某种单位由 $f(1)$增至 $f(n)$，这时称该算法的空间代价是 $f(n)$。

算法的时间代价（或称时间复杂性）：当问题规模以某种单位由 1 增至 n 时，对应算法所耗费的时间也以某种单位由 $g(1)$增至 $g(n)$，这时称该算法的时间代价是 $g(n)$。

在这里有 3 个值得注意的概念：问题的规模、空间单位和时间单位。问题的规模比较容易理解，一般根据问题本身的性质合理地确定。例如对 n 个记录进行排序，这里 n 即可