面向21世纪课程教材

面向21世纪课程教材
Textbook Series for 21st Century

算法与数据结构

——C语言描述

张乃孝 主编
陈 光 刘 筠 韩玉真 编

高等教育出版社
HIGHER EDUCATION PRESS

面向21世纪课程教材
Textbook Series for 21st Century

算法与数据结构

——C语言描述

张乃孝 主编
陈 光 刘 筠 韩玉真 编

高等教育出版社
HIGHER EDUCATION PRESS

图书在版编目（CIP）数据

算法与数据结构——C语言描述/张乃孝主编；陈光，刘筠，韩玉真编. —北京：高等教育出版社，2002.9
ISBN 7-04-011221-3

Ⅰ.算… Ⅱ.①张…②陈…③刘…④韩…
Ⅲ.①电子计算机-计算方法②数据结构③C语言-程序设计 Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2002）第068778号

算法与数据结构——C语言描述
张乃孝 主编 陈光 刘筠 韩玉真 编

出版发行	高等教育出版社	购书热线	010-64054588
社　　址	北京市东城区沙滩后街55号	免费咨询	800-810-0598
邮政编码	100009	网　　址	http://www.hep.edu.cn
传　　真	010-64014048		http://www.hep.com.cn
经　　销	新华书店北京发行所		
排　　版	高等教育出版社照排中心		
印　　刷	中国农业出版社印刷厂		
开　　本	787×960　1/16	版　　次	2002年9月第1版
印　　张	22	印　　次	2002年9月第1次印刷
字　　数	400 000	定　　价	23.00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。

内 容 提 要

本书系统地介绍算法与数据结构方面的基本知识。全书共分九章：第一章绪论，从问题求解引入数据结构和算法的基本知识和抽象数据类型的概念，是全书的综述；第二章至第五章以介绍基本数据结构为主线，重点讨论了线性表、串、栈、队列、树和二叉树等各种结构的抽象模型、存储表示和运算的实现，同时也通过一些实例，讲述了这些结构的应用；第六章至第八章重点介绍各种重要的检索、排序算法和图论中的典型算法，同时也对字典、散列表、最佳二叉排序树、平衡的二叉排序树、B树、B+树、堆、有向图、无向图和网络等重要数据结构概念、表示方法和基本操作展开了系统的讨论；第九章是本书的最后一章，对算法的分析与设计技术做了论述，从算法分类的角度对全书的算法做了总结。

本书内容体系完整，概念清楚，论述充实，取材得当，面向实际应用，可以作为高等院校理工科本科相关专业的“算法与数据结构”或者“数据结构”课程教材。对于有一定C语言程序设计基础的理工科学生，讲授全部内容大约需要50学时至70学时。对于生、化、医、农等专业的本科生和理工科的专科生，建议在教学时跳过目录中部分或全部带“*”的章节。

前　言

20世纪70年代初，著名的计算机学者E. W. Dijkstra首先提出了把程序设计看作一门科学的观念，揭开了程序设计"革命"的新篇章。从那时起，兴起了一股围绕程序设计理论研究的热潮。开始时人们都把注意力集中在程序的构造和分析上，更确切地讲，就是集中在程序文本所表述的算法结构上。然而不久之后人们便认识到：要对大型复杂程序的构造进行系统而科学的研究，必须同时对这些程序中所包含的复杂数据结构进行系统的研究。事实上，程序就是在数据的某些特定结构和表示方式的基础上对抽象算法的描述。如果不了解施加于数据上的算法，就无法决定如何构造数据。反之，算法的选择往往在很大程度上依赖于作为基础的数据结构。简而言之，如何选择算法与数据结构是程序设计中两个相互联系的不可分割的问题。著名的计算机学者N. Wirth用"算法＋数据结构＝程序"这个公式表达了算法与数据结构的相互关系及在程序设计中的地位。

本书以"算法与数据结构"为名，旨在系统地介绍算法与数据结构方面的基本知识，希望读者阅读本书后，能够在程序设计的实践中运用这些知识，编制出正确且有效的程序。全书共分九章：第一章绪论，从问题求解引入数据结构和算法的基本知识和抽象数据类型的概念，是全书的综述；第二章至第五章以介绍基本数据结构为主线，重点讨论了线性表、串、栈、队列、树和二叉树等各种结构的抽象模型、存储表示和运算的实现，同时也通过一些实例，讲述了这些结构的应用；第六章至第八章重点介绍各种重要的检索、排序算法和图论中的典型算法，同时也对字典、散列表、最佳二叉排序树、平衡的二叉排序树、B树、B＋树、堆、有向图、无向图和网络等重要数据结构概念、表示方法和基本操作展开了系统的讨论；第九章是本书的最后一章，对算法的分析与设计技术做了论述，从算法分类的角度对全书的算法做了总结。

书中各章的最后有小结、习题和少量的上机实习题；对于非基本要求的内容均在目录中添加了"＊"；正文后面还提供了三个附录，分别给出C语言的基本知识、学习本书内容所必备的数学基础和一份学生上机实习报告样本，供教学时参考。

"算法与数据结构"在许多高等院校中是一门公共必修课，北京大学把它作为第一批理科各系主干基础课列入学校的教学计划之中。通过讲授本课，使学生较全面地理解算法和数据结构的概念，掌握各种数据结构和算法的实现方式，比较不同数据结构和算法的特点。由于它是一门理论与实际紧密结合的课程，因此，要求

学生在课堂学习的基础上，完成适当的上机实习。通过实习过程，使学生能够真正提高利用计算机解决实际问题的能力。

本课程要求学生已修过“计算概论”课程，掌握一种程序设计语言的基本知识，并且具有一些使用程序设计语言进行编程的经验。考虑到目前国内计算概论的教学大多结合C语言进行，而且用C语言描述算法十分有效，分析使用C语言描述的算法又比较直观，因此，本书采用C语言作为描述语言，并且假设读者都具备了这方面的基础知识。

本书的编写大纲在广泛地征求北京大学理科各系的要求和意见的基础上，由张乃孝执笔。初稿的第二章至第五章由陈光执笔，第六章至第八章由刘筠执笔，附录A和附录B由韩玉真执笔，其余部分由张乃孝执笔。从2000年开始，本书作为北京大学“算法与数据结构”课程的教材在校内许多院系试用，得到广大授课老师和学生的支持与鼓励。特别要感谢谢柏青、丁文魁、刘从义、裘宗燕、罗静初、刘楚雄、赵俊峰、王洪庆、刘剑飞和赵丹群等老师以及北京大学1999级和2000级的许多同学，他们对本书内容提出了许多宝贵的意见和建议。在他们的支持下，两年中三易其稿，三次在北京大学校内印刷。全书修改工作在张乃孝的主持下完成，韩玉真、陈光和刘筠也参加了部分整理工作。尽管如此，错误和疏忽之处在所难免，衷心希望广大读者批评指正。

本书可以作为高等院校本科和专科教材。根据我们的实践，对于有一定C语言程序设计基础的理工科学生，讲授全部内容大约需要50学时至70学时；对于生、化、医、农等专业的本科生和理工科的专科生，建议跳过目录中部分或全部带“*”的章节。

书中的主要算法和数据结构的定义以及部分授课的胶片可以在以下网址找到：

http://www.math.pku.edu.cn/teachers/zhangnx/ds/

或者

ftp://162.105.69.121/zhangnx/ds/

张乃孝

2002年8月于北京大学

znx@pku.edu.cn

目　录

第一章 绪 论

计算机应用技术的飞速发展，正在逐步揭开计算机神秘的面纱，把它从高等学府和研究院的专业实验室中解放出来，成为人们工作、学习和生活中不可缺少的工具。时至今日，使用计算机进行文字处理、网上通信、休闲娱乐等已经成为一种时尚。然而，对于计算机工作原理的无知，必然大大束缚了使用者的手脚。尤其是对于那些希望使用计算机解决自己的某些特殊问题或者希望尝试使用某种新方法去解决某些问题的“高级用户”，了解计算机的工作原理、学习程序设计的基本方法，依然是十分重要的。

计算机科学研究的重点是信息在计算机中的表示和处理问题。这些问题出现在许多不同的研究层次和不同的应用领域。从程序设计的观点看，信息在计算机中的表示就是“数据结构”研究的问题；信息在计算机中的处理就是“算法”研究的问题。因此，学习算法和数据结构的基本知识是了解计算机工作基本原理、掌握程序设计基本技术的必经之路。

本章首先从一个具体应用的例子出发，讨论使用计算机求解的大概过程，从中引出数据结构和算法的有关问题，然后介绍算法的分析和抽象数据类型这两个重要的概念，最后指出学习本课程应该注意的一些问题。

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。

为一个实际问题开发一个程序系统通常可以分成以下四个阶段：

1. 分析阶段。这个阶段的任务是弄清所要解的问题是什么；并且把它用一种语言（自然语言、说明语言或数学语言）清楚地描述出来。

2. 设计阶段。这个阶段的任务是建立程序系统的结构，重点是算法的设计和数据结构的设计。对于大型的复杂的程序系统，这一阶段往往还包括模块的设计。

3. 编码阶段。它的主要任务是根据设计的要求，采用适当的程序设计语言（C语言、C++语言或 Java 语言），编写出可执行的程序。

4. 测试和维护。其任务首先是发现和排除在前几个阶段中产生的错误，经测试通过的程序便可投入运行，在运行过程中还可能发现隐含的错误和问题，因此，还必须在使用中不断地维护和完善。

显然，与本课程关系最为密切的为设计阶段，但由于设计阶段介于分析和编码这两个阶段之间，因此，讨论中也牵涉到它们。下面，通过一个很小的实际问题，重点讨论从问题的分析到算法和数据结构的设计、精化过程。

1.1.1 问题分析

考虑一个多叉路口(见图 1.1)，在这个路口中，共有五条道路相交。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，而如果分组数越少，则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。

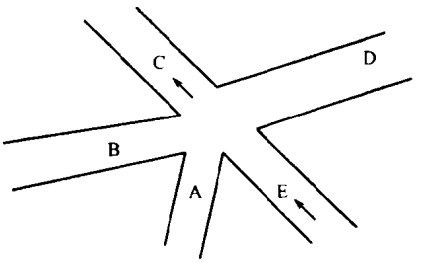

图 1.1 一个交叉路口的模型

根据这个路口的实际情况，可以确定 13 个可能通行的方向：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。有些方向很明显不能同时进行，如 A→B 与 B→C 这两条路线就不能同时行驶等。为了叙述方便，下面把 A→B 简写成 AB，并且用一个小椭圆把它框起来，在不能同时行驶的路线间画一条连线(表示它们互相冲突)，便可以得到图 1.2 所示的图式。这样得到的表示可以称为“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。在本书的第八章将讨论这种结构。)

上面这样做就把要解决的问题借助图的模型清楚而严格地表达出来，结果是把一个实际问题变成了另一个抽象问题。这个问题就是：要求将图 1.2 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。显然，这个问题的解是不惟一的，最容易的方法就是把每个结点代表的一条行驶路线都单独分成为一组，一共

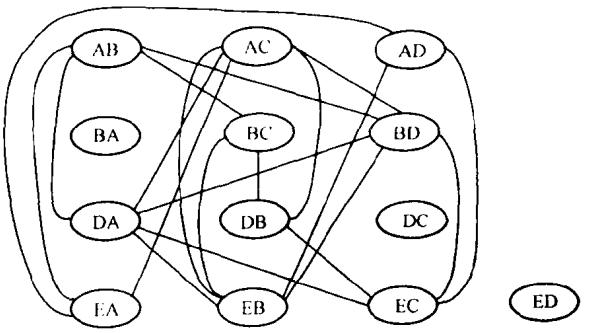

图 1.2 交叉路口的图式模型

分成 13 组，就得到这个问题的一个“可行解”。进一步地，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”。

由于上面的分析过程用了图的抽象表示，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化、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的颜色都不相同。

1.1.2 程序设计

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着色方案，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(显然只有一种着色方式)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的组合数，可能的数目很大：例如，两个组按 $1:n-1$ 分配，就有 $C_n^1 = n$ 种可能，按 $2:n-2$ 分配，就有 $C_n^2 = n(n-1)$ 种可能，等等。当考察了所有放在两个组里可能的组合后都不可行的话，接着考虑分为三组、四组的各种组合。

由此可见，这类穷举法对结点少的问题(称为“规模小的”问题)还可以使用；对规模大的问题，则会由于求解时间会随着实际问题规模的增长而呈指数性上升，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，根据处理顺序的不同，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

这个解告诉人们，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。

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，只要一个问题可以被化为着色问题）就可以用这个程序求解。

为此需要做以下工作：首先，为问题中所有相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，编写程序，并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，那么程序的编写就非常简单。假设需要着色的图是 G，集合 V1 包括图中所有未被着色的结点，着色开始时 V1 是 G 所有结点的集合（用 G.V 表示）。NEW 表示已确定可以用新颜色着色的结点的集合。

从 V1 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；
```

本程序片段执行后，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 V1 为空（下面用 isSetEmpty(V1) 表示）。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（可以用一个变量 color 计数）。这个程序片段里涉及对集合和图的操作，包括由集合 V1 中去掉一个元素 v（表示为 removeFromSet(V1,v)），向集合 NEW 里增加一个元素

v(用 addToSet(NEW,v)表示),检查结点 v 与结点集合 NEW 中各结点之间在图 G 中是否有边连接(用 notAdjacentWithSet(NEW,v,G)),等等。如果有了这些结构和操作,程序的实现可以非常直接地用下面的程序给出:

```
int colorUp(Graph G)
{  int color = 0;
   V1 = G.V;
   while(! isSetEmpty(V1))
   {  emptySet(NEW);
      while ( ∃v∈ V1 && notAdjacentWithSet(NEW,v,G))
            {  addToSet(NEW,v);
               removeFromSet(V1,v);
            }
      ++color;
   }
   return(color);
}
```

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,通常只提供了一些基本数据类型(如整数、实数、字符等)和一些数据构造手段(如数组、记录、指针等)。要解决这个计算问题,用户必须自己用选定的语言所提供的机制实现这些结构(集合、图等)和操作(对集合元素的增删、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的过程中,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地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,人们重点关心的是下面三个要素:

第一,这个结构中各元素之间的(抽象)逻辑关系,在有的教材中简称为**逻辑结构**。例如,线性表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。除了线性表外,字符串、栈和队列都是线性结构。与线性结构对应的是

非线性结构，如树形结构、图结构等都是非线性结构。

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，线性表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；如果线性表中各元素是通过指针连接存放在内存，就称为链接结构（表示）。除了上述两种存储结构外，还有索引结构（表示）、散列结构（表示）等。

第三，这个结构具有的行为特征。这主要体现在与这个结构相关的运算的定义上。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，通常所称的“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；而其对应的所谓“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。

下面，对本书中将要讨论的各种主要结构做一简单介绍。

(1) **线性表** 线性表是一种逻辑上十分简单但应用非常广泛的数据结构，线性表中各元素之间是一种简单的“线性”关系。顺序表和链表是两种常用的实现线性表的数据结构，它们也是许多复杂结构的基本表示形式。

(2) **字符串** 字符串也是一种特殊的线性结构，它以字符为元素。本书中讨论的字符串，主要采用顺序表示的形式，因此，每个字符可以直接访问。

(3) **栈与队列** 栈和队列是两种十分重要的数据结构。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；而队列实行先进先出的原则，最先到达的元素也最先离开队列。

(4) **树与二叉树** 树和二叉树都属于树形结构，在逻辑上表示了结点的层次关系，是一种非线性结构。在树中，每个元素可以有多个后继，但最多只能有一个前驱，许多实际的和理论的问题中都可以抽象成某种树形结构。二叉树中每个元素最多只能有两个后继，并且二叉树中每个元素的后继都被区分为“左”或“右”，哪怕只有一个后继也是如此。由于二叉树和一般树林（树的有序集合）之间存在一种一一对应关系，所以，使二叉树的研究具有特别重要的意义。

(5) **字典** 字典可以看作是一种二元组的集合，每个二元组包含着一个关键码和一个值。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。按关键码进行检索是字典中最常用的操作。根据对字典中进行元素插入、删除频率的不同，有所谓“动态字典”和“静态字典”之分。“散列表”是实现字典的有效结构，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。由于树形结构便于灵活地执行插入和删除，所以经常用来表示动态字典。

(6) **图** 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，边集合中的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。

要对数据结构的概念给出一个严格的定义是比较困难的。在参考文献[2]中，笔者根据区分数据结构的三要素，给出了一种五元组的定义。而在更多的文献中，则简单地用数据元素的集合和元素之间关系的集合来定义数据结构，这种二元组的定义强调了数据的逻辑结构，而忽视了数据的存储结构和相关操作的特点。因此，又在需要时引入数据结构的“表示”方式，专指一种逻辑结构可以有不同的存储结构；另外，用数据结构的“实现”专指在具体数据结构的表示基础上各种操作的具体过程描述。

在数据结构的讨论中重点研究的是“结构”，而把组成结构的那些元素抽象成一个“结点”。结点是数据结构中的基本单位，当然，在实际应用中一个结点可以是一个字符、一个整数，也可以是一个结构。例如，在讨论某班学生的成绩单时，可以采用线性表作为逻辑结构，每个学生的成绩作为线性表中的一个元素，但这个元素中可能包括学号、姓名和各课程的具体成绩等。

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。一个算法可能有若干个输入，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被明确描述，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注入了青春和活力，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。有了计算机的帮助，使得许多过去仅靠人工无法计算的大量复杂问题有了解决的希望。不过，使用计算机进行计算，首先要解决的是如何将被处理的对象存储到计算机中，也就是要选择适当的数据结构。本书第六章讨论的检索是一类最常用的算法，然而由于被检索的数据采用了不同的表示方式而引出许多复杂程度差别很大的检索算法；而在第七章对一个最简单的排序问题，同样是用数组的顺序表示，就给出许多思想方法不同的排序算法。在本书中，算法的讨论贯穿始终。从简单的顺序表的插入、删除，到复杂的求带权图所有结点之间最短路径，都离不开算法。

在实际应用中，算法的表现形式千变万化，但是也和数据结构类似，许多算法的设计思想具有相似之处，因此，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。

二分法检索就是采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）中找一个整数，要求查出这个整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，将该元素与所找整数进行比较，如果所查整数比较小，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，若所查整数较大，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，就能很快确定待查整数的存在与否，并确定如果存在的话它在什么位置。

还有一类常用算法被称为**回溯法**。有一些问题，需要通过彻底搜索所有可能情况而寻找一个满足某些预定条件的最优解。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，其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。

动态规划法与分治法相似，都是把一个大问题分解为若干较小的子问题，通过求解子问题而得到原问题的解。不同点是：分治法每次分解的子问题数目比较少，子问题之间界限清楚，处理的过程通常是自顶向下进行；动态规划法分解的子问题可能比较多，而且子问题相互包含，为了重复使用已经计算的结果，要把计算的中间结果全部保存起来，通常是自底向上进行。

与回溯法相似，**分枝界限法**（branch and bound）也是一种在表示问题解空间的树上进行系统搜索的方法。所不同的是，回溯法使用了深度优先策略，而分枝界限法一般采用广度优先策略或者采用最大收益（或最小损耗）策略，同时还利用最优