数据结构与算法

〔美〕A·V·阿霍　J·E·霍普克罗夫特　J·D·厄尔曼　著

科学出版社

数据结构与算法

A.V. 阿霍
〔美〕J. E. 霍普克罗夫特 著
J.D. 厄尔曼

唐守文　宋俊京　陈　良　李海军　译

洪加威　校

科 学 出 版 社

1 9 8 7

内 容 简 介

本书系统地讲述数据结构和最基本的计算机算法，全书共十二章：前三章介绍程序设计的基本概念，包括算法、抽象数据类型和基本数据结构；第四、五两章介绍以集合为基础的若干抽象数据类型及实现方法；第六、七两章介绍图算法；第八章介绍各种整序算法；第九、十两章概括了算法设计技巧和算法分析方法，简要介绍了分治法，动态规划、贪婪法和回溯法等设计算法的基本思想及算法分析中常见的递归方程的解法；最后两章讨论与存贮管理有关的算法和数据结构．本书每章末均附有大量难易程度不等的习题．

本书可作为计算机软件专业高年级大学生和研究生的教材，也适于其他从事计算机研究和应用的人员参考．

A. V. Aho J. E. Hopcroft J. D. Ullman
DATA STRUCTURES AND ALGORITHMS
Addison-Wesley, 1983

数据结构与算法
A. V. 阿霍
〔美〕J. E. 霍普克罗夫特 著
J. D. 厄尔曼
唐守文 宋俊京 陈 良 李海军 译
洪加威 校
责任编辑 那莉莉
科学出版社出版
北京朝阳门内大街137号
北京景山学校印刷厂印刷
新华书店北京发行所发行 各地新华书店经售

1987年10月 第一版 开本：787×1092 1/32
1987年10月第一次印刷 印张：16 8/8
印数：0001—8,100 字数：373,000
统一书号：15031·872
本社书号：5315·15-8
定价：3.85元

前　言

本书介绍的数据结构和算法是当今计算机程序设计的基础。本书主要取材于我们的早期著作《计算机算法的设计与分析》一书的前六章，此外还补充了外存贮器算法和存贮器管理算法方面的内容。这就使得本书比较适合作为数据结构和算法方面初级课程的教材。作为预备知识，我们仅仅要求读者熟悉一种程序设计的高级语言，例如Pascal语言。

我们力求使本书概括计算机程序设计中广泛采用的数据结构和算法。在表述与实现算法时，我们非形式地使用了抽象数据类型。尽管抽象数据类型在一些程序语言中刚刚出现，但是我们认为，无论在什么语言中，抽象数据类型都是进行程序设计的有力工具。

我们还介绍了如何分析程序的运行步数和时间复杂度，并将这种分析看作是解决问题的过程中不可缺少的一部分。这反映了我们坚持已久的观点：随着计算机速度的不断提高，有待解决的问题的规模也必将越来越大，因而随着硬件的更新换代，算法的时间复杂度的意义不仅不会被削弱，而且会越发显示其重要性。

算法的表述

我们按照Pascal语言的习惯来表述算法和数据结构，因为Pascal语言是广为人们熟知的。当然，也不难用其它任何一种高级语言来实现我们给出的算法。从问题的提出，到可以运行的计算机程序的完成，了解这样一个解决问题的全过程是十分重要的。因此在本书的前几章，我们不仅给出抽象算法，

也给出具体的Pascal程序。

本书用法

第一章是导论，解释了“从问题到程序”的求解过程，以及在这一过程中抽象数据类型所起的作用，还介绍了程序的运行步数及大“O”和大“Ω”的概念。

第二章介绍表、栈、队以及反映函数这一数学概念的抽象数据类型——映射。第三章介绍树以及实现树操作的若干有效的数据结构。

在第四、第五两章中介绍的若干重要的抽象数据类型都是以集合为数学模型的。我们将深入讨论字典和优先队列，并介绍一些实现的方法。例如二叉搜索树、偏序树、检索树以及2-3树等等。更深入的讨论是在第五章。

有关图的内容在第六、七两章。第六章是有向图，第七章是无向图。从这两章开始，我们的讨论重点将从数据结构转向算法，当然也还要介绍一些表示图的基本数据结构。我们将给出一些重要的图算法，包括先深搜索算法、求最小生成树算法、求最短道路算法以及极大匹配算法。

第八章是一些重要的内部整序算法，包括快速整序、堆整序、箱整序以及一些非常简单（虽然不很有效）的整序算法。本章还给出了求中位数，或求其它顺序统计量的一些线性时间算法。

第九章讨论了递归过程的渐近分析方法。当然包括递归方程的建立以及求解递归方程的技巧。

第十章介绍一些重要的算法设计技巧。包括分治法、动态规划法、局部搜索法以及各种形式的树搜索法。

最后两章涉及外存组织和存贮器管理。第十一章讨论外部整序和大规模外存组织，包括B-树和索引结构。第十二章

涉及存贮器管理，分别讨论了固定分块或可变分块以及采取显式或隐式垃圾回收等四种不同要求的管理技术。

本书作者在加利福尼亚大学、康奈尔大学和斯坦福大学曾用本书中的内容作为大学生和研究生的数据结构和算法课程的教材。例如在斯坦福大学，高年级大学生和一年级研究生的数据结构课程曾用本书的一个简写本作为为时十周的教材，其中包括一～四、九、十、十二各章，以及五～七章的一部分内容。

练习题

本书的各章后面都附有一部分不同深度和难度的练习题，其中许多习题用来检验对本章内容的掌握情况。一些较难的习题标有星号*，还有一些更难的习题标有双星号**，这些习题适用于更高级的教程。各章末的文献注记介绍了有关进一步阅读的参考资料。

致谢

我们要感谢贝尔实验室提供的便利，他们杰出的UNIX™系统上的制版及数据通讯设备使我们能够顺利地完成手稿。我们的许多同事阅读了手稿，他们提出了很有价值的意见和建议。特别要感谢Jon Bentley, Brian Kernighan, Steve Mahaney, Kerry Nemovicher, Paul Niamkey, Rob Pike, Bob Tarjan和 Peter Weinberger，他们的建议给予我们很大帮助。最后，我们要向Claire Metzger夫人表示最热诚的谢意，感谢她在手稿的加工与排版方面所给予的帮助。

A.V.阿霍

J.E.霍普克罗夫特

J.D.厄尔曼

目　录

第一章　算法的设计与分析

要用计算机解决一个具体问题，编写程序的工作就要经历这样一些步骤：首先要使问题有一个简明、严格的提法，然后设计求解方法，使这种解法能为计算机实现，还要经过测试和文本编制，最后还要评价解法的优劣。本章将概括地介绍我们在这些步骤中所采取的方法。程序设计中常用的一些算法和数据结构将在后面几章讨论。

1.1　从问题到程序

许多问题的最初提法常常既不精确又不简练。还有一些问题，譬如说，如何维护世界和平？如何配制一份使品尝家满意的食谱？这些问题也许不可能简单而精确地表述成能为计算机解决的形式。即使是我们认为能用计算机求解的问题，也常常需要在很大的范围内确定问题的参数，而那些合理的参数值只有通过实验才能确定。因此，要用计算机解决问题，必须首先以简明的、严格的方式将问题表述清楚。可以说，成功的关键在于明确要解决的问题。

如果能用一个形式模型来刻划问题的某些方面，那么这样做是很有益的。一旦将问题形式化，我们就可以依据这个严格的模型对问题求解。对于形式化了的问题，我们容易知道是否已有现成的程序可以利用。即使没有现成的程序可用，至少我们可以利用这个形式模型所具有的种种性质来构造好的解法。

数学或其它科学中的几乎所有分支都可以作为某一类具

体问题的抽象模型。例如，在数值计算问题中常用的数学模型为线性方程组(用于求解电路的电流强度，或者求解梁架结构中的应力等)或微分方程(用于预报人口增长情况，或求解化学反应速率等)；符号与文本处理问题中常用字符串及形式语法作为模型，这类问题包括编译(从程序设计语言到机器语言的程序翻译)、信息检索（譬如从图书馆目录中查找某些特定文字)等等。

算法

对问题建立了适当的数学模型以后，就可以依据这一模型求解。我们最初的目标是要给出一个算法形式的解法。所谓算法就是一个由若干条指令组成的有穷序列，其中的每一条指令的含义都是明确的，且每条指令的执行次数以及每次执行的时间也都是有限的。例如，常见的整数赋值语句$x:=y+z$就可以作为一条指令，只要进行有限次的逐位相加及进位动作就可以执行完这条指令。在一个算法中，有些指令可能需要重复执行多次，因此指令的执行次数可能远远超过指令的条数。但是我们要求，无论对于什么样的输入，算法中任何一条指令都不能被无穷多次地执行。也就是说，对于任何输入，算法在执行了有限多次指令后一定要终止。因此，一个程序如果对任何输入都不会陷入无限循环，那么它就是一个算法。

关于算法的上述定义有一点需要说明。我们说过，算法中的每一条指令都必须具有“明确的含义”，并且每条指令的执行时间必须是有限的。但是，人们对于同一句话可能会有不同的理解。一个人认为是“明确的含义”未必能使另一个人确切地理解。即使我们能确切地理解每一条指令的含义，但要证明一个指令序列对于任何输入都能终止也常常是困难

的。然而，关于一个指令序列是否构成一个算法，我们总有可能通过论证和反证而取得一致看法。如果有人宣布他找到了一个算法，那么他就必须证明那确实是一个算法。在1.5节中，我们要讨论如何估计一个计算机程序的运行时间，如果我们能得到程序运行时间的一个上界，就证明了程序的执行时间是有限的，因而是一个算法。

本书除使用Pascal语言外，还常用一种伪语言表述算法。伪语言是将某种程序设计语言的基本结构与非形式的英文或中文叙述语句相结合的产物。我们的伪语言采用了Pascal语言的基本结构，但是几乎任一种常用的程序设计语言都可以代替Pascal来表达我们的算法。下面举例说明我们在编写计算机程序的过程中所采取的基本步骤和一般方法。

例 1.1 考虑交叉路口交通信号灯的设计问题。我们需要一个这样的程序：它以路口上许可通行的所有转弯路线（直行通过的路线也看作一种转弯）作为输入，程序将这些转弯分组输出，使得同一组内的转弯互不冲突。为了使路口的信号管理简单有效，我们还要求使转弯分组的组数尽可能少。这样，如果用信号控制器的每一相位指挥一组内的转弯，那么我们就能以最少的相位数有效地指挥路口的交通。

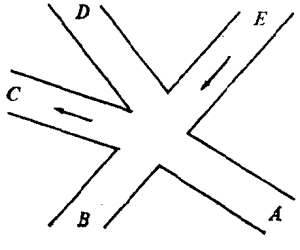

图 1.1 一个交叉路口

图1.1是一个相当复杂的交叉路口的图示。它是位于普

林斯顿大学附近的JoJo's酒吧间门前的交叉路口。人们知道，这里的交通很不方便，尤其往返这个路口时就更困难。图中的C和E是两条单行路线，其它路线都可以双向行驶，因此这里共有13种不同方向的转弯路线。其中有些转弯是互不冲突的，例如AB(从A到B)和EC就是这样。另外还有些转弯是互相冲突的，例如AD和EB，因此不允许它们同时通行。这个路口的交通信号灯必须禁止象AD与EB这样的冲突转弯同时通行，但是应允许象AB与EC这样的无冲突转弯同时通行。

首先要找一个适当的抽象模型将问题形式化。我们可以用一个称为图的数学结构作为这个问题的模型。一个图由一些点与一些连接点的线所组成，通常称这些点为**顶点**，称这些线为**边**。我们可以用一个具有13个顶点的图来刻划这个路口的交通情况：其中的每一个顶点代表一种转弯；如果某两个顶点所代表的转弯是相互冲突的，则在这两个顶点之间连接一条边。这样得到的图如图1.2所示。图1.3中的表格是这个图的另一种表示方法。如果表中第i行与第j列所对应的两个顶点之间有边相连，则在表中第i行、第j列的位置填写1。

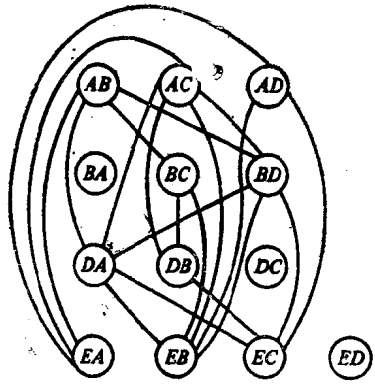

图 1.2 表示转弯冲突的图

	AB	AC	AD	EA	BC	BD	DA	DB	DC	EA	EB	EC	ED
AB					1	1	1			1			
AC						1	1	1		1	1		
AD										1	1	1	
EA													
BC	1							1			1		
BD	1	1					1				1	1	
DA	1	1				1					1	1	
DB		1			1							1	
DC													
EA	1	1	1										
EB		1	1		1	1	1						
EC			1			1	1	1					
ED													

图 1.3 转弯冲突表

利用这样的图，我们可以将交通信号灯的设计问题形式化为图的着色问题. 所谓**图着色**就是要求对图中的每个顶点着染一种颜色，并使得任意两个有边相连的顶点具有不同的颜色. 不难看出，我们的问题就是要用尽可能少的几种颜色对上面表示冲突转弯的图进行着色.

图着色问题的研究已有几十年的历史了，人们对这个问题的算法理论也作了大量的研究. 但是，如何用尽可能少的颜色对任意给定的图进行着色的问题，是人们称为“NP-完全问题”的一大类问题中的一个. 关于这类问题的所有已知算法基本上都采用“穷试法”. 对于图着色问题而言，“穷试法”就是先用一种颜色着色；如果行不通，再用两种颜色尝试所有可能的着色方式；如果还不行，再试用三种颜色……；如此做下去，直到找出符合要求的着色方式为止. 如果考虑得细致，也可以使上述过程稍微加快一点. 然而人们普遍相信，尽管上述算法是一种再明显不过的方法，但是要对它进行本质性的改进也许是不可能的.

由于图着色问题的计算量太大，因此产生了寻求最优解

的可行性问题。对此有三种解决方法。如果图很小，可以用穷试法寻求最优解，这样可使问题得到彻底解决。但是对于较大的图，无论怎样提高程序的效率，穷试法的计算量都太大以致根本无法实现。第二种方法是对需要着色的图进行具体分析，找出它的一些特点，这些特点也许能使图具有某些特殊性质，使我们不必穷试所有可能，即可求得最优解。第三种方法是将问题稍加修改，把要求放宽一点，只要求令人满意的解，而不一定要求最优解。这样也许能有一个快速算法，使得它对较小的图的结果很接近最优解。由于一般交叉路口的情况不会比图1.1所示的情况更复杂，所以我们就采用第三种方法求解。所谓的"启发式方法"就是可以快速求得满意解（不一定是最优解）的一种方法。

以下的"贪婪"算法就是解决图着色问题的一种合理的启发式方法：首先用第一种颜色给尽可能多的顶点着色，然后用第二种颜色给尽可能多的尚未着色的顶点着色，如此作下去。对于每一种新颜色，我们按下述步骤进行着色：

1．选一个尚未着色的顶点，给它着以新色。

2．逐个检查所有尚未着色的顶点，对每一个这样的顶点，查看是否有边将它与某个已着新色的顶点相连。如果没有这样的边，则给该顶点着以新色。

这种方法之所以称为"贪婪法"，是由于它每遇到一个可着新色的顶点就着色，而不考虑这样做是否会妨碍其它更多的顶点着新色的可能性。事实上，如果在某些情况下不要太"贪婪"，而是留下某个本来可以着新色的顶点，那么这种新颜色就能给更多的顶点着色。例如在图1.4所示的图中，假设顶点1已着红色，如果按顶点标号的次序执行"贪婪"算法，则只能再给一个顶点2着红色。但是如果留下顶点2不着红色，则还可以给3和4两个顶点着红色。

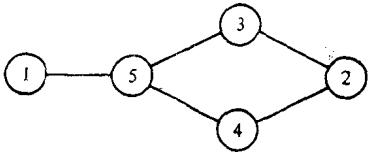

图 1.4 图例

我们以图1.2为例说明贪婪算法。假设从AB开始执行，首先给AB着蓝色，然后再给AC，AD和BA着蓝色，因为这四个顶点中任意两个顶点之间没有边相连；但是BC不能着蓝色，因为AB与BC之间有边，同理，BD，DA及DB也不能着蓝色，因为它们都至少与一个已着蓝色的顶点相连；然而DC可着蓝色，后面还有EA，EB及EC都不能着蓝色，而ED可以着蓝色。

再开始用第二种颜色，譬如给BC着红色，BD也可着红色，但是DA不行，因为BD与DA之间有边；同理DB也不能着红色；DC已着过蓝色；EA可着红色；剩下的其它未着色顶点都有边与某红色顶点相连，所以它们都不能再着红色。

这时还有DA，DB，EB及EC是尚未着色的。如果将DA着为绿色，那么DB也可着绿色；而EB及EC则不能着绿色，这两个顶点可以用第四种颜色着色，譬如用黄色。最后，着色的情况总结为图1.5中的表。表的中间一栏是贪婪算法对转弯冲突图的着色结果。同一颜色的那些转弯是可以无冲突地同时通行的，此外还有一些可以同时通行的附加转弯路线。这些转弯在表中右边一栏里列出，它们与左边一栏中的转弯不冲突，并且也可以通过贪婪算法求出。当交通灯允许一种颜色的转弯通行时，它也可以让这些附加转弯安全通行。

用贪婪算法得到的图着色方案不一定是最优的，也就是说，它所用的色数不一定是最少的。对于它所产生的着色结

颜色	转弯路线	附加转弯路线
蓝色	*AB*, *AC*, *AD*, *BA*, *DC*, *ED*	—
红色	*BC*, *BD*, *EA*	*BA*, *DC*, *ED*
绿色	*DA*, *DB*	*AD*, *BA*, *DC*, *ED*
黄色	*EB*, *EC*	*BA*, *DC*, *EA*, *ED*

图 1.5 图1.2中图的着色结果

果，我们可以用算法理论评价其优劣。在图论中，所谓k-团是指一个具有k个顶点的图，其中的任意两个顶点之间都有边相连。显然，要对一个k-团进行图着色必须使用k种颜色才行，因为其中的任意两点都不能具有相同的颜色。

在图1.2内，AC，DA，BD与EB这四个顶点组成了一个4-团。由此可见，少于四种颜色的着色方案是不可能的，用我们原来的话说，不可能用少于四个相位的信号控制器管理图1.1所示的交叉路口。因此，我们在这里用贪婪算法得到的着色方案，在使用尽可能少的颜色的意义下是最优的。

这样，根据图1.5中的表，我们可以设计一个具有四个相位的信号灯控制器，使每一相位对应表中的一行。当控制器处在某一相位时，信号灯允许相应一行中的所有转弯通行，同时禁止其它转弯路线通行。这样的控制器所用的相位数不可能再减少了。 □

伪语言与逐步加细

一旦给问题建立了适当的数学模型，我们就可以依据这个模型严格地表述算法。一开始，算法的雏形常常是用普通的叙述语句表达的，这样的叙述语句必须逐步修改成更详细、更精确的指令。例如，在前面关于图着色的贪婪算法的叙述中，有这样一句话："选一个尚未着色的顶点"。我们希望这种叙述的含义是足够明确的，以便读者能够把握我们的意