计算机算法设计与分析

卢开澄　谭明术　编著

中国铁道出版社

计算机算法设计与分析

卢开澄　谭明术　编著

中　国　铁　道　出　版　社

1998年·北京

(京)新登字 063 号

内 容 简 介

计算机算法设计与分析是计算机专业一门十分重要的专业基础课，本书提供了一本十分实用的教科书。

本书重点介绍了算法设计的基本方法，如优先策略、分治策略、动态规划以及 DFS 和 BFS 等搜索法。还分别针对具体的排序、查找，匹配，概率算法，几何算法，数论算法，近似算法和 NP 理论等问题作了介绍。

本书适用于计算机，数学及相关专业作教材使用，也可作为广大从事计算机软件开发的人员自修用书。

图书在版编目(CIP)数据

计算机算法设计与分析/卢开澄，谭明术编著．—北京：中国铁道出版社，1997.12

ISBN 7-113-02909-4

Ⅰ.计… Ⅱ.①卢… ②谭… Ⅲ.①电子计算机-算法设计②电子计算机-算法分析 Ⅳ.TP301.6

中国版本图书馆 CIP 数据核字(98)第 02674 号

书　　名：计算机算法设计与分析
著作责任者：卢开澄　谭明术
出版·发行：中国铁道出版社(100054，北京市宣武区右安门西街 8 号)
策划编辑：严晓舟
责任编辑：严晓舟
封面设计：马利
印　　刷：北京市燕山联营印刷厂
开　　本：787×1092　1/16　印张：10.5　字数：258 千
版　　本：1998 年 9 月第 1 版　1998 年 9 月第 1 次印刷
印　　数：1—2000 册
书　　号：ISBN7-113-02909-4/TP·287
定　　价：16.20 元

摘　要

算法设计与分析是计算机科学重要理论课程。本书用深入浅出的方法讨论有关的基本算法，以及若干重要问题。如优先策略、动态规划、分治策略及搜索法等。在讨论算法的同时对若干重要的问题作比较深入的剖析，如 FFT，分类查找问题等。本书可作为计算机专业教材。有关的专业也可用作参考书。

序　言

本书力图写成一本计算机系高年级适用的算法教材。充分考虑学生的接受能力，以便学生掌握计算机算法设计与分析的基本概念和方法。

计算机贵在神速。如每秒能做数亿次乘法的大型机，相当于集中了数亿人的计算能力，的确身手不凡。计算机的发展速度之快也是惊人的。短短50年时间内，它不仅完全改变了自己，也改变了这个世界面貌。在今天，人们几乎无处不感觉到它的存在，世界也从此进入了计算机时代。计算机元器件从继电器，真空管，晶体管，集成电路，大规模集成电路，发展到超大规模集成电路；结构上从每台机器作为运算工具的“单兵作战”模式，发展到今天将范围遍及各大洲的计算机群组成的网络；算法也从单机的串行计算发展到网络上并行的分布计算。

一台作10^9次/秒运算的计算机的效率超过10亿人同时工作，它可以作中期的天气预报，可以控制庞大的化工厂的生产过程，甚至可以驾驭现代化战斗机器，……，它看似无所不能。然而，事实上并非如此。举个极为简单的例子：要搜索所有26个英文字母组成的排列，以每年365天计，每秒能完成10^9个排列的超高速电子计算机不停地工作，需要$26!/(365\times24\times3600\times10^9)\approx1.2\times10^{10}$年。即使计算机运行速度随着技术的提高而提高，恐怕这是也难以实现的，因为计算机速度提高也是有极限的。

由此可以看出，计算机的能力不是无限的，有些问题理论上能实现而实际上办不到。讨论哪类问题可由计算机来作，哪类问题计算机则不能作，属于可计算性理论研究的范畴。本课程只研究那些理论上是可计算的一类问题的算法，并对算法进行分析。

一个算法就是一系列的将输入转换为输出的计算步骤。

算法设计与分析这门学科还很年轻，尚未定型，还在发展过程中，但无疑是十分重要的。

本书介绍了算法设计的四个基本方法，即优先策略、分治策略、动态规划以及DFS与BFS的搜索法。还分别针对具体的排序、查找、匹配、概率算法、几何算法、数论算法、近似算法和NP理论等问题也作了介绍，以引起读者进一步学习和研究的兴趣。

以实例来熟悉算法是本书作者的意愿，希望读者通过它来较快掌握和理解相应算法。

对于算法的描述，本书采用自然语言按步骤叙述的办法，而不采用类似于某一高级语言较形式化的方式，作者认为这样做比较自然，灵活。有些算法的思想本来很直观，一旦加以形式化，反而将简单的问题异乎寻常地复杂化了。

算法的研究本属于数学范畴，特别是算法的复杂性分析，更是离不开数学，但算法这门学科也不全是纯数学问题，它不能不考虑相应的数据结构。研究算法离不开数据结构，本书十分强调这个观点。

本书适用于计算机、数学及相关专业作教材使用。

由于作者水平有限，缺点和不妥之处请读者不吝指正。

作　者

1997年

目　录

第一章　基础知识

§1.1 引　　言

计算机是通过程序来完成各种操作的。什么是程序呢？有人说

程序＝算法＋数据结构。

这是有一定道理的。它至少说明了两点：一是要让计算机完成任务首先要有算法；其次，研究算法不能不考虑数据结构。

算法(algorithm)一词来自19世纪一位波斯数学家，它就是计算的步骤和法则，也就是一系列的将输入转换为输出的计算步骤。

在计算机出现的30年代，人们就研究过算法并使其形式化，同时提出并研究了各种形式的计算模型，证明了“停机问题”的不可计算性。所谓停机问题指的是：是否有算法对于任意给定的程序，都能判断它是否造成停机？从理论上判定什么样的问题可以通过算法利用计算机求解，什么问题不可以，属于“可计算性理论”研究的课题。虽然可计算性理论对于计算机科学有着重要的影响，但在理论上知道一个问题在计算机上可解还不足以知道实际上是否办得到。这类例子很多。

有许多具有实际应用的问题是可以通过计算机解决的，但解决这些问题所需时间和存储空间是如此之大，以致于它实际上无法运行。显然，程序运行所需时间和空间的分析具有极重要的意义。因此，这个问题已经成为计算机科学领域研究的一个重要课题，叫做复杂性分析，或时间和空间复杂性分析。

对算法的研究，第一步是设计算法，第二步是复杂性分析。对同一问题，若有两种不同的算法，两种算法孰优孰劣要依复杂性分析的结果来判断。需要的时间较短，存取单元较少的算法是较好的。

对一个算法的评价一般从以下四个方面进行考察：

1. 正确性；
2. 运行时间；
3. 占用空间；
4. 简单性；

所谓正确性就是指在给定有效的输入之后，算法经过有限时间的计算并产生正确答案。要对算法正确性进行证明，最有效的办法就是数学归纳法。运行时间是指一个算法在计算机上运算所花费的时间，它大约等于计算机执行一种简单操作(如赋值，运算，比较操作等)所需时间乘以算法中简单操作次数。占用存储空间通常考虑三个方面的占用情况：一是存储算法本身占用，二是输入输出数据所占用，三是算法在运行过程中临时占用。算法运行过程中临时占用的存储空间大小被定义为算法的空间复杂性，用数量级的形式给出。简单性指的是程序实现复杂

程度。

§1.2 算法分析

算法分析是指对一个算法所需资源进行预测。资源常指计算时间。本书主要采用单处理器——随机存储器(RAM)作为计算模型,算法可用程序表达。在RAM模型中,指令一条接一条地执行,无并行操作。

输入规模(input size)依赖于研究的具体问题,在许多问题中,指的是输入的元素个数。

例1 下面以求两个n阶方阵$A=(a_{ij})_{nxn}$,$B=(b_{ij})_{nxn}$之积为例,令$C=A\times B=(c_{ij})_{nxn}$,其中

$$c_{ij}=\sum_{k=1}^{n}a_{ik}b_{kj} \qquad i,j=1,2,\cdots,n$$

试用自然语言形式表达两个n阶方阵的乘积如下:

S1. i←1,C←**0**。

S2. 若i≤n则做

始 j←1,转S3 **终**,否则转S7。

S3. 若j≤n则做

始 k←1,转S4 **终**,否则转S6。

S4. 若k≤n则做

始 $c_{ij}\leftarrow c_{ij}+a_{ik}b_{kj}$,k←k+1,转S4 **终**

否则转S5。

S5. j←j+1,转S3

S6. i←i+1,转S2

S7. 输出(c_{ij}),结束。

其中C←**0**表示对矩阵C置零,即

$$c_{ij}\leftarrow 0,\quad i,j=1,2,\cdots,n。$$

始和**终**为语句括号,如S2,当i≤n时做两件事,即j←1,和转S3。

若i≤n则依次做两个操作:j←1和转S3。

时间复杂性分析:矩阵$C=(c_{ij})_{nxn}$有n^2个元素,每个元素要作n次乘法,n次加法,故需n^3次乘法,n^3次加法。所以时间复杂性为$\Theta(n^3)$。

空间复杂性分析:$3n^2$个存储单元用以存放3个矩阵A,B,和C。故空间复杂性为$O(n^2)$。能否用少于$3n^2$个单元?留给读者思考。

后面§3.2节读者将了解求两个n阶方阵之积的时间复杂性为$\Theta(n^3)$并非天经地义。

算法研究的第一步是针对问题设计算法,进一步则必须对所设计的算法进行理论分析,看一看究竟所设计的算法是否可行,跟其它的算法比较好坏程度。比如下面的汉诺(Hanoi)塔问题便是一例。该问题是组合数学名题。

问题(汉诺塔):n个圆盘依其半径大小从下而上套在柱A上,如图1.1所示。每次只允许取一个转移到B或C柱子上,而且不允许大的放在小的上方。现A柱上有大小不等的n个圆盘,要求将它转移到C柱上,试问应如何进行,并对移动的盘次进行估计。限制只有A,B,C三根柱子可供使用。

第一步算法设计：

n＝2 时，先将最上面一个盘子取出套在 B 柱上，再将第二盘套在 C 柱上，最后将 B 柱上的盘取下套在 C 柱上。两个盘子的转移结束。

n＝3 时，先利用上面 n＝2 的办法，将前面两个盘子套在 B 柱上，再将 A 柱上最后一个盘子套上 C 柱上，最后再利用 n＝2 时的算法，将 B 柱上的两个盘子套到 C 柱上，3 个盘子时转移结束。

n 个盘子的汉诺塔问题可以递归地利用上述的算法。n＝2，n＝3 已解决。

假定 n－1 的汉诺塔问题已解决，先将 A 柱上前面 n－1 个盘子套在 B 柱上，再将 A 柱上最后一个盘子取下套到 C 柱上，最后再利用 n－1 个盘子的办法，将 B 柱上 n－1 个盘子套到 C 柱上。n 个盘子的问题结束。

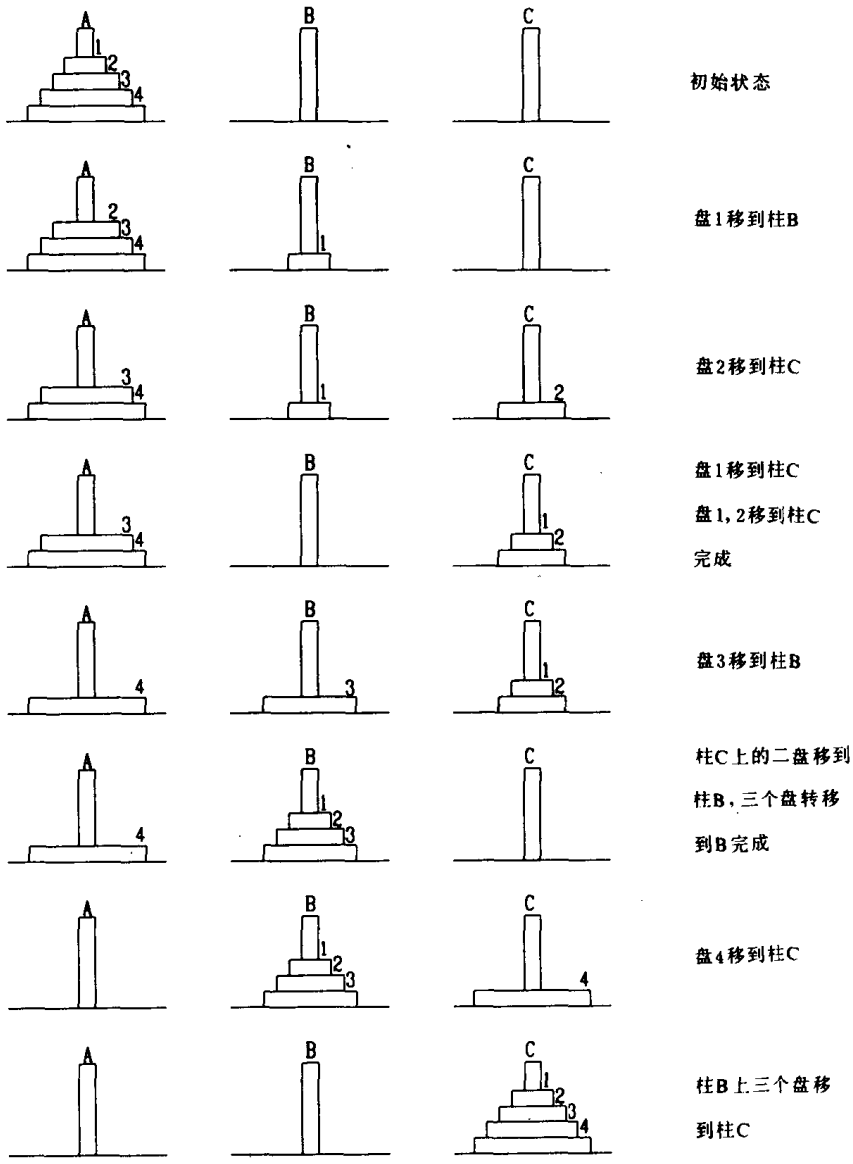

图 1.1

算法分析

令 S_n 表 n 个盘子的汉诺塔问题搬动的盘次，则有

$$S_n=2S_{n-1}+1$$

$$S_1=1$$

$$S_2=2S_1+1=2+1$$

$$S_3=2S_2+1=2^2+2+1$$

$$S_4=2S_3+1=2^3+2^2+2+1$$

$$\cdots$$

$$S_n=2^{n-1}+2^{n-2}+\cdots+2+1=2^n-1$$

所以汉诺塔问题是指数型问题，$S_n=2^n-1$ 是 n 的指数函数。

由于我们对一个算法的评价是以输入规模 n 的数量级来衡量，当 n 很大时低阶项相对来说不太重要，有时还忽略最高次项的常系数，说它是 nk 或 2^n 量级。所以在算法分析时，通常把算法中包含简单操作次数的多少就作为算法的时间复杂性。

例 2 设 A 是含 n 个数据项的一个数组。$A=(a_1,a_2,\cdots,a_n)$。对某一指定项 x，如果 x 在此数组中，设 $x=a_k$，则输出它的下标 k；如果 x 不在其中，则输出 0 作为结束。

顺序搜索算法(Sequential Search)

输入 A，n，x。A 是一个具有 n 项的数组。

S1. j←1

S2. 若(j≤n)∧(A[j]≠x)则做

　　始 j←j+1，转 S2 终，否则转 S3

S3. 若 j>n 则做

　　始 j←0，转 S4 终否则转 S4，

S4. 打印 j，结束。

平均情况分析，设 I_i 表示要找的数 x 在数组 A 中第 i 个位置(即 $x=A[i]$)的情况，$i=1,2,\cdots,n$，I_{n+1}表示 x 不在数组中的情况。又设 t(I)是算法在输入 I 时所做比较次数。显然，对于 $1\leqslant i\leqslant n$，$t(I_i)=i$，$t(I_{n+1})=n$。设 p 是 x 在数组中的概率，若 x 确定在该数组中则 $p=1$。并假设 x 出现在每个位置的可能性是相同的，于是对 $1\leqslant i\leqslant n$，$p(I_i)=\frac{p}{n}$，$p(I_{n+1})=1-p$。

所以总的比较次数

$$\begin{aligned}t(n)&=\sum_{i=1}^{n+1}p(I_i)t(I_i)\\&=\sum_{i=1}^{n}\frac{p}{n}i+(1-p)n\\&=p\frac{n+1}{2}+(1-p)n\end{aligned}$$

若肯定 x 在数组中，这时 $t(n)=\frac{n+1}{2}$，即平均要查找一半数组 A。

最坏情况：当 x 是数组中最后一项和 x 不在数组中时，$T(n)=n$。即是说 x 要和所有的 n 个项进行比较。最好的情况，一次比较找到。

上面讨论的数组 $A=(A[1],A[2],\cdots,A[n])$是无序的，所以只能采取顺序查找的办法，

即 A[1],A[2],…,A[n]依次比较的方式。如若数组 A 是按顺序排列，即

$$A[1]<A[2]<\cdots<A[n]$$

则顺序查找显然是效率比较低的一种办法，可以用二分查找法，即取序列 A 的中间元素 $A\left[\left\lfloor\frac{n}{2}\right\rfloor\right]$ 与 x 作比较，若 $x=A\left[\left\lfloor\frac{n}{2}\right\rfloor\right]$ 则一次成功，若

$$x<A\left[\left\lfloor\frac{n}{2}\right\rfloor\right]$$

则 x 必在 $A[1],A[2],\cdots A[\frac{n}{2}-1]$ 中。又若

$$x>A\left[\left\lfloor\frac{n}{2}\right\rfloor\right]$$

则 x 可从序列

$$A\left[\left\lfloor\frac{n}{2}\right\rfloor+1\right],A\left[\left\lfloor\frac{n}{2}\right\rfloor+2\right],\cdots,A[n]$$

中去找。作一次比较，可将搜索省去一半。而且对后面的查找可继续递归地利用二分查找的办法。最坏情况下比较的次数将大大减少。

下面再以这例子说明数据结构对研究算法的重要性。二分查找对于静态的序列 A，以顺序存储为宜，即 A[i]与 A[i+1]相邻二元素的储存地址也是相邻的。这样二分查找计算地址比较容易。问题在于实际上数据 A 总是动态的，不断地有新的元素插入到序列中去，也有元素从序列中删除，所以顺序存储这样一种数据结构实际上是不实用的。一个元素的插入或删除都将引起将近一半的元素要“搬家”，即改变存放的地址。后面将详细地讨论“查找”问题，它必须仰仗于提出新的数据结构。这里只不过先强调一下研究算法不能不考虑它的数据结构。二分算法适用顺序存储这种方式，但顺序存储这样一种数据结构实际上很不适用。

本书用数学方法对算法的效率进行分析，而不考虑用什么样的计算机，或用什么样的计算机语言。运算时间 T(n)是输入规模 n 的函数。本书中讨论的函数有多项式，对数函数，指数函数，或它们的乘积。设机器每步花时间为 10^{-6} 秒。现举例列表比较于后。

表 1.2.1

T(n) \ n	10	20	40	80
lnn	2.3×10^{-6}秒	3×10^{-6}秒	3.7×10^{-6}秒	4.4×10^{-6}秒
n	10^{-5}秒	2×10^{-5}秒	4×10^{-5}秒	8×10^{-5}秒
nlnn	2.3×10^{-5}秒	6×10^{-5}秒	14.7×10^{-5}秒	35×10^{-5}秒
n^2	10^{-4}秒	4×10^{-4}秒	16×10^{-4}秒	64×10^{-4}秒
n^{10}	2.8 小时	118.5 天	3.3×10^{2} 年	3.4×10^{3} 世纪
2^n	10^{-3}秒	1 秒	12.7 天	3.8×10^{3} 世纪
3^n	5.9×10^{-2}秒	0.97 小时	3.9×10^{3} 世纪	4.7×10^{22}世纪

从表 1.2.1 可以看出，它们关于输入规模 n 的增长速度的差别是非常明显的。效率分析是非常重要的，我们可用以判断应用某一算法解某一类问题时，允许的规模有多大，还可以判定哪一个方法较好。

§1.3 常用记号

下面介绍本书经常用到的一些符号和术语。

$\mathbf{N}$　　非负整数集合{0,1,2,…}

$\mathbf{N}^+$　　正整数集合{1,2,…}

$\mathbf{R}$　　实数集合

$\mathbf{R}^+$　　正实数集合

$\mathbf{R}^*$　　非负实数集合

$|S|$　　有限集合 S 中元素个数

$|x|$　　字符串 x 的长度,即 x 中字符个数

$\lfloor x \rfloor$　　小于或等于 x 的最大整数,简称 x 的向下取整(floor of x)

$\lceil x \rceil$　　大于或等于 x 的最小整数,简称 x 的向上取整(ceiling of x)

设 f 和 g 是由 N 到 R^* 的两个函数,引进下列渐近记号。

定义 1　令 $f:\mathbf{N}\to\mathbf{R}^*$

$O(f)\triangleq\{g:\mathbf{N}\to\mathbf{R}^* \mid \exists\ c\in\mathbf{R}^*,\exists\ n_0\in\mathbf{N},\forall\ n\geqslant n_0,$使 $g(n)\leqslant cf(n)\}$也就是 O(f)为函数 $g:\mathbf{N}\to\mathbf{R}^*$ 的集合,它以 cf(n)为上界。即若存在 $c\in R^*$,且 $\lim\limits_{n\to\infty}\frac{g(n)}{f(n)}=c$,则 $g\in O(f)$。

基于上述概念,例若 $f(n)=n^3,g(n)=n^2$,则 $g\in O(f)$,但 $f\notin O(g)$。

定义 2　令 $f:N\to R^*$

$\Omega(f)\triangleq\{g:\mathbf{N}\to\mathbf{R}^* \mid \exists\ n_0\leftarrow\mathbf{N},\exists\ c\in\mathbf{R}^*,\forall\ n\geqslant n_0,$使 $g(n)\geqslant cf(n)\}$

若 $\lim\limits_{n\to\infty}\frac{g(n)}{f(n)}=c>0$,c 可为无穷大,则 $g\in\Omega(f)$。

定义 3　令 $f:\mathbf{N}\to\mathbf{R}^*$。

即 $\lim\limits_{n\to\infty}\frac{g(n)}{f(n)}=c,c\in\mathbf{R}^+$,则 $g\in\Theta(f)$。其中 $c\in\mathbf{R}^+$ 表示 $c\neq0,c\neq\infty$。$g\in\Theta(f)$意味着 g 和 f 同阶。$\Theta(1)$表示一个常数。

O,Ω,Θ 有以下性质(假定 $f,g,h:1\mathbf{N}\to1\mathbf{R}^*$):

(1)　$f\in O(g),g\in O(h)\Rightarrow f\in O(h)$;

(2)　$f\in O(g)\Leftrightarrow g\in\Omega(f)$;

(3)　$f\in\Theta(g)\Leftrightarrow g\in\Theta(f)$;

(4)　$O(f+g)=O(\max\{f,g\})$。

§1.4 递　　归

在一个算法中,常常含有对其自身的调用,当我们在分析该算法的时间复杂性时,就会出现递归关系式。所以下面对如何求解递归式作简要介绍。

首先,有些递归式可化为一个和来求解,其次用生成函数来求解,这两种方法容易在许多书中可找到,如拙著《组合数学》一书。

还是以 §1.2 的汉诺塔问题为例,已知

$$S_n=2S_{n-1}+1$$

$S_1=1$，

前面已介绍

$$S_2=2S_1+1=2+1=3$$

$$S_3=2S_2+1=2(2+1)+1$$

$$=2^2+2+1$$

$$S_4=2S_3+1=2^3+2^2+2+1$$

$$\cdots\cdots$$

假定 $S_n=2^{n-1}+2^{n-2}+\cdots+2+1=2^n-1$

可证 $S_{n+1}=2S_n+1=2^{n+1}-2+1=2^{n+1}-1$。

下面介绍母函数法。设

$$S(x)=S_1+S_2x+S_3x^2+\cdots$$

$$x:\ S_2=2S_1+1$$

$$x^2:\ S_3=2S_2+1$$

$$\vdots$$

求和得 $$S(x)-S_1=2xS(x)+(x+x^2+\cdots)$$

$\therefore$ $$(1-2x)S(x)=S_1+\frac{x}{1-x}=1+\frac{x}{1-x}$$

$$=\frac{1}{1-x}$$

$$S(x)=\frac{1}{(1-x)(1-2x)}=\frac{A}{1-2x}+\frac{B}{1-x}$$

$$=\frac{A(1-x)+B(1-2x)}{(1-2x)(1-x)}$$

比较等式两端分子：$(A+B)-x(A+2B)=1$

$\therefore$ $$A+B=1$$

$$A+2B=0$$

$\therefore$ $$B=-1,\quad A=2$$

即

$$S(x)=\frac{2}{1-2x}-\frac{1}{1-x}$$

$$=2[1+(2x)+(2x)^2+\cdots]-(1+x+x^2+\cdots)$$

$S(x)$的x^{n-1}项系数为S_n，

$\therefore$ $$S_n=2^n-1$$

另一方面，由于假设

$$S(x)=1+S_2x+S_3x^2+\cdots$$

$$-)\quad 2xS(x)=\quad 2S_1x+2S_2x^2+\cdots$$

$$(1-2x)S(x)=1+(S_2-2S_1)x+(S_3-2S_2)x^2+\cdots$$

由于 $$S_2-2S_1=S_3-2S_2=\cdots=S_n-2S_{n-1}=1$$

$\therefore$ $$(1-2x)S(x)=1+x+x^2+\cdots$$

$$(1-2x)S(x)=\frac{1}{1-x}$$

$$S(x)=\frac{1}{(1-x)(1-2x)}$$

异途同归得了相同的结果。求得 $S(x)$，便可求得序列 $S_1,S_2,\cdots,S_n,\cdots$。

§1.5 图

客观世界中有许多状态用图来描述，可以达到直观，便于思考的效果。一个图是由一些顶点和连接两顶点间的连线组成，而不计较连线长度、弯曲及顶点位置。

定义 1 一个图 G 是一个二元组(V(G),E(G)常简记为 G=(V,E)。

图中每条边均为无向边的图称为无向图，如图 1.5.1(a)

图中每条边均为有向边的图称为有向图，如图 1.5.1(b)

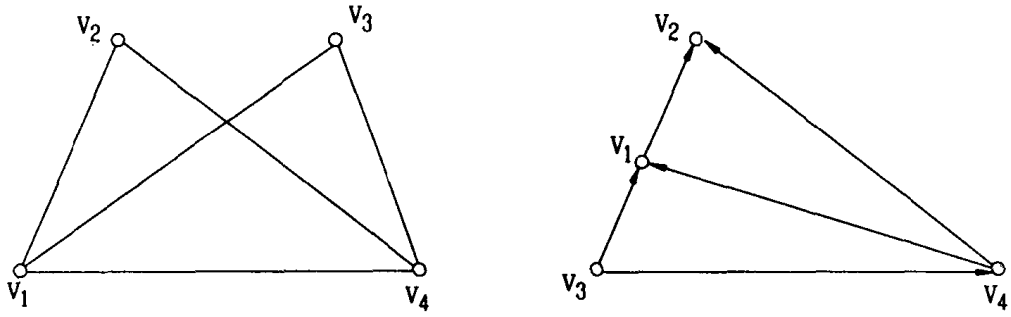

图 1.5.1

若顶点间有一边相联，则称这两点相邻接；若顶点 v 是边 e 的一顶点，则称 v 与 e 关联。与顶点 v 相关联的边数叫顶点 v 的度，记为 d(v)。

定义 2 设图 G=(V,E)及图 G′=(V′,E′)。对于∀ v∈V,∃ v′∈V′,∀ e∈E,∃ e′∈E′,若存在一一映射 $g:v_i\leftrightarrow v_i',e\leftrightarrow e'$，且 $e=(v_i,v_j)$是 G 的一条边，当且仅当 $e'=(g(v_i),g(v_j))$是 G′的一条边，则称 G 与 G′同构，记作 G≃G′。

由定义知：若 G 与 G′同构，其充要条件是两个图的顶点和边分别存在着一一对应，且保持关联关系。

例如：在图 1.5.2 中(a)与(b)同构，(c)与(d)同构。

定义 3 给定图 G=(V,E)。设 $v_0,v_1,\cdots,v_n\in V,e_1,e_2,\cdots,e_n\in E$，其中 e_i 是关联于顶点 v_{i-1},v_i 的边 i=1,2,……,n。称点边交替序列：

$v_0e_1v_1e_2\cdots e_nv_n$ 为联接 v_0 到 v_n 长途为 n 的一条路径。

路径的始点 v_0 等于终点，即 $v_0=v_n$ 时，这路径称为圈，也叫回路，v_0ev_0，即边 e 的两端点相同时，称为自环。

顶点不同的路径叫路。路上边的数目称为路径长度。

定义 4 给定图 G=(V,E)，另有一图 H=(V′,E′)，若 V′⊆V,E′⊆E，则称 H 是 G 子图。

定义 5 若无向图 G 中任意两点均有一条路相联，则称 G 为连通图。对于有向图 G=(V,E)，去掉边的方向后的图若是连通的，则称有向图 G 是连通图。

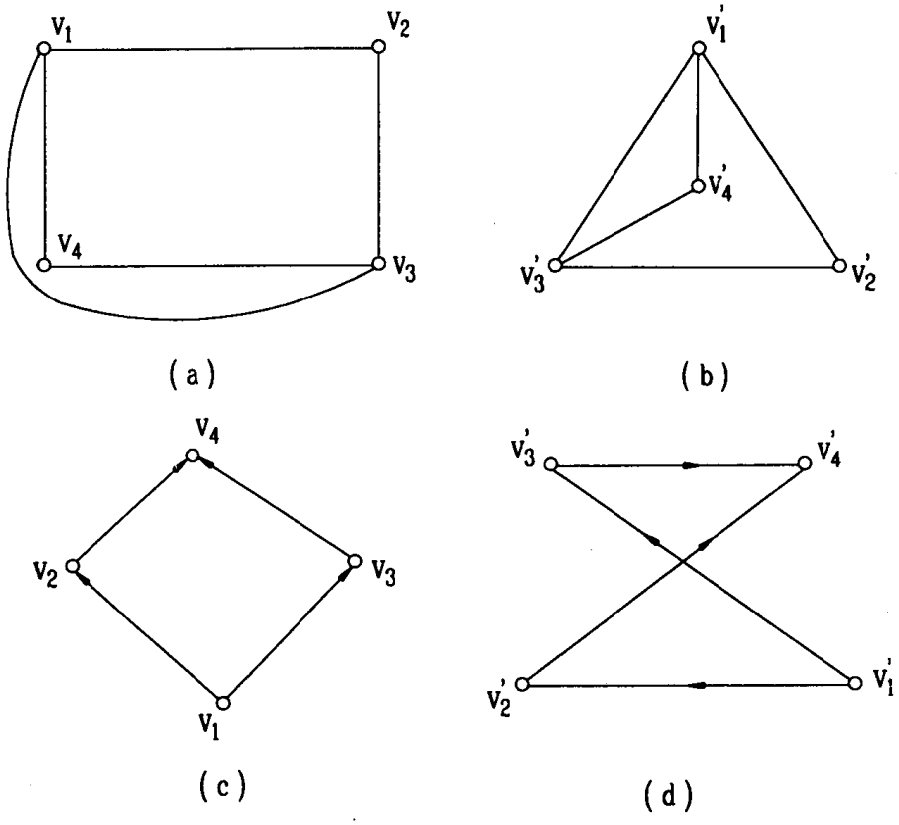

图 1.5.2

定义 6 在有向图 G=(V,E)中,若从顶点 u 到 v 存在一条有向路,则称从 u 可达 v。

若两点之间要么只有一条边相联,要么没有边相联,且任何边的两顶点不重合,即不存在自环,则称这样的图为简单图。以后无特别申明,我们所指的图为简单图。

定义 7 设图 G=(V,E)为一简单图。$V=\{v_1,v_2,\cdots,v_n\}$,把 $A(G)=(a_{ij})_{nxn}$称为 G 的邻接矩阵,其中

$$a_{ij}=\begin{cases}1 & \text{边}(v_i,v_j)\in E\\ 0 & \text{否则}\end{cases}$$

例如图 1.5.3 中的邻接矩阵为

$$A(G_1)=\begin{bmatrix}0&1&1&1&1\\1&0&1&0&0\\1&1&0&1&0\\1&0&1&0&1\\1&0&0&1&0\end{bmatrix},\quad A(G_2)=\begin{bmatrix}0&1&0&1\\0&0&0&1\\1&1&0&1\\0&0&0&0\end{bmatrix}$$

定理 1 设 A(G)是图 G 的邻接矩阵,则$(A(G))^k$ 中的第 i 行,第 j 列元素 $a_{ij}^{(k)}$等于 G 中联接 v_i 与 v_j 的长度为 k 的路的数目。

这里$(A(G))^k$ 是邻接矩阵 A(G)的 k 次方,即$(A)^k=\underbrace{A\cdot A\cdot\cdots\cdot A}_{k个}$以 k=2 为例 A·A=$A^2=(a_{ij}^{(2)})$,$a_{ij}^{(2)}=a_{i1}a_{1j}+a_{i2}a_{2j}+\cdots+a_{in}a_{nj}$,若 $a_{i1}=a_{1j}=1$,即 $v_i\to v_1\to v_j$ 存在一条从 v_i 两步到 v_j