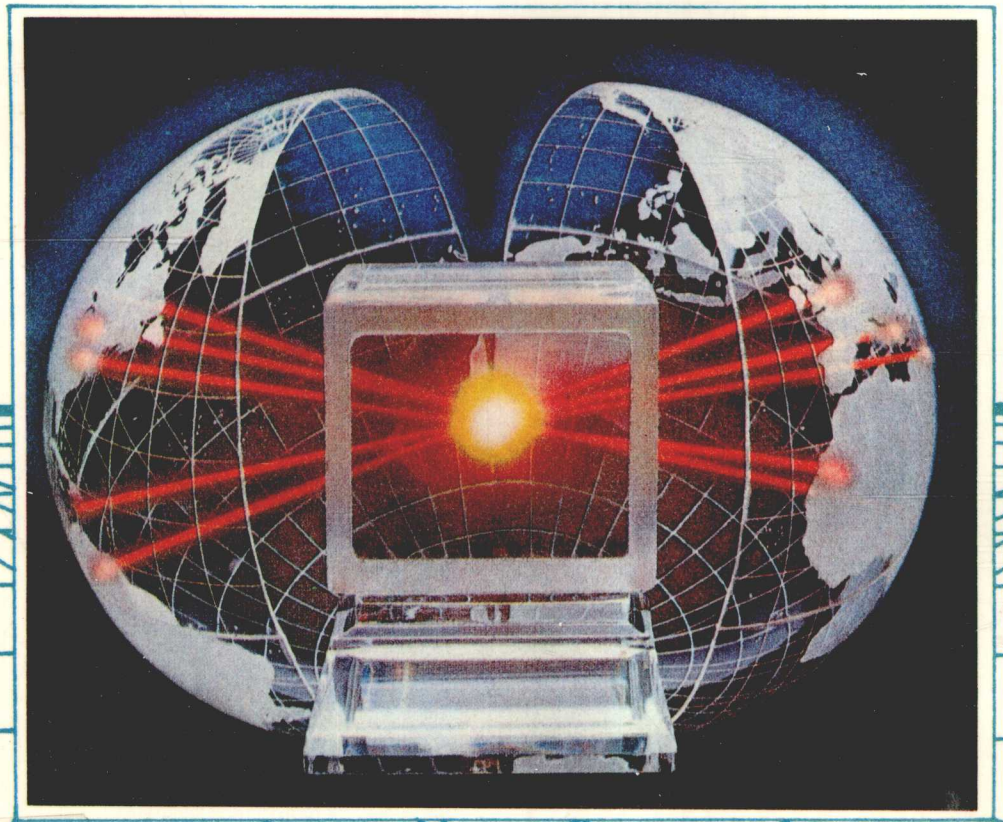
数据结构原理

刘大有　李岳峰　编著

数据结构原理
数据结构原理
数据结构原理



吉林大学出版社

数据结构原理

刘大有　李岳峰　编著

吉　林　大　学　出　版　社

内容提要

本书系统介绍了数据结构的概念、原理和技术。主要包括：线性表，数组的存储和字符串的运算，树形结构，图，排序与查找操作，内存管理，文件的结构和组织方式。本书概念清楚，内容丰富，对于每一种数据结构以及所采取的运算和操作，都给出了详细的算法和时间复杂性分析，并对其中一些有代表性的算法给出了正确性证明。为了培养学生，应用数据结构技术解决实际问题的能力，书中还给出了许多例题和习题。

本书可作为高等院校计算机软件专业和计算机及应用专业的教材或参考书，也可供广大从事研究和使用计算机的科技人员学习参考。

数据结构原理

刘大有　李岳峰　编著

责任编辑、责任校对：王瑞金　　封面设计：张沐沉

吉林大学出版社出版　　吉林大学出版社发行

（长春市东中华路 29 号）　　吉林农业大学印刷厂印刷

开本：787×1092 毫米　1/16　　1994 年 10 月第 1 版

印张：16.625　　1994 年 10 月第 1 次印刷

字数：421 千字　　印数：1—4200 册

ISBN 7—5601—1622—1/TP・33　　定价：12.00 元

前　言

1972 年，霍尔（C. A. R. Hoare）在“数据结构札记”一文中澄清了关于数据结构的术语和概念等方面所存在的杂乱局面，特别阐明了：不了解施加于数据上的算法就不知道怎样去构造数据，反之，若不深入研究作为其基础的数据结构，则算法的优美结构与设计将无从谈起。总之，算法与数据结构是一对不可分割的孪生兄弟。

我们正是从上述观点出发对本书内容进行组织的，这主要表现在：简洁叙述了计算算法时间复杂性的方法。对书中的算法都给出了时间复杂性分析，并注意了分析的严格性。此外，还给出了典型算法之正确性的证明。

其次，在本书中我们特别使用了一种被称之为 ADL 的算法描述语言，其主要原因在于：比之 D. E. Knuth（克努斯）的算法描述语言，ADL 能方便地描述递归；比之类 PASCAL 语言，ADL 避免了过多涉及数据类型的定义，书写起来较为简便；ADL 既便于算法之定量分析，又便于在较高抽象级别上勾划出算法的轮廓。

最后，本书还增加了一些新内容，例如：计算有向图强连通分支的快速算法，静态树，重量平衡树等等。新增内容使有关章节之内容更加完善。

面向典型数据结构的算法设计（这是我们对《数据结构原理》一书的看法！）是一门基于恰当选择的数据结构的关于算法的构造性技术。我们期待读者从我们仔细选择并细致描述的典型例子中，获得或初步获得如下能力：

- 针对实际问题在较高抽象级别上，确定数据结构并构造算法；
- 针对实际问题选择合适的数据结构并设计结构优良的算法；
- 对算法的时、空复杂性进行分析，并针对具体问题对两者进行恰当权衡。

由于计算机科学技术的飞速发展，参考资料、时间和水平等方面的限制，书中难免有诸多不足和错误，为此特别期望广大读者给予批评指正。

作　者

1994. 08. 25

目　录

第一章 绪 论

本章特简述数据结构与算法的关系，介绍有关数据、数据结构和算法的概念，扼要说明了全书各章的主要内容.

§1 引 言

1967 年，唐·欧·克努特（D. E. Knuth）在他的《计算机程序设计技巧》一书中，提出了“计算机科学是有关算法的学问”，并指出“对所有的计算机程序设计说来，算法的概念总是最基本的”. 1969 年，C. A. R 霍尔（C. A. R. Hoare）的论文“计算机程序设计公理化基础”和 1972 年 E. W. 戴克斯特拉（E. W. Dijkstra）的论文“结构程序设计札记”，都集中研究由程序正文所代表的算法的结构. 然而，对程序构造进行系统而科学的研究，首先是对包含复杂数据集的大型复杂程序而言的. 因此，程序设计的方法学必然包含数据结构的所有方面. 实际上，程序就是在数据的某些表示方式和结构的基础上对抽象算法的具体表达. 霍尔通过“数据结构札记”一文明确阐明了，程序的构成与数据结构是两个不可分割地联系在一起的问题. 1976 年，N·沃思（N. Wirth）旗帜鲜明地表达了这一观点，他用“算法＋数据结构＝程序”作为其专著的题目.

正是基于上述观点，本书把“算法分析基础”做为一章，并且对书中施加于数据结构上的几乎所有算法都给出了时间复杂性分析，对书中典型数据结构之上的算法又都给出了算法正确性的证明. 这充分体现了本书从数据结构与算法是不可分割的基点上对数据结构展开讨论的宗旨.

计算机发展初期，主要用于数值计算，处理的对象是数值数据，这类数据具有结构简单、形式统一且数据量小等特点. 但是，随着计算机科学、技术的发展，其应用领域不断扩大，特别是人类步入信息社会的今天，一方面计算机面对的数据具有结构极其复杂、数据量巨大且形式多样化的特点，另一方面它在大多数应用中存取大量数据的能力被认为是最重要的特征. 这些都充分证明了，要设计出高效、准确、适应性强和易读的程序，对数据的性质和数据元素间的关系进行深入研究已成为首要的任务. 换言之，必须对数据内部的结构，以及计算机内如何表示、存储和有效操作这种结构的方法与技术，有系统、透彻的了解.

本书主要阐述各种数据结原理与技术. 全书共分十一章. 第一章主要介绍有关数据结构和算法的基本概念. 第二章对算法评价标准和算法分析进行了讨论. 第三章至第六章论述了各种基本数据结构的特点、存储方式和基本运算. 若干重要例子将用来说明相关技术的多种多样的应用. 第七章讲解排序操作. 排序又称之为分类、整检或整序. 就是把数据元素（如记录）按关键词整序，比如排成递增次序. 第八章讲述查找操作. 对于给定的关键词，找出含有该关键词的那个记录. 该章还阐述了插入和删除运算. 第九章主要讲述了内存管理的各种基本策略. 第十章和第十一章分别论述了外排序的基本方法和文件基本组织结构.

§2 数据结构概念

直观地说，数据系指一些事实，或指一批数，或者指一个符号集合，等等．我们把组成数据的“事实”、“数值”或“符号”称之为数据元素．数据元素是组成数据的基本单位．数据是计算机程序使用和加工的“原料”．例如，一个简单的数值计算程序所使用的数据是一些实数或整数，一个编译程序使用和加工的数据是源程序．又如，一个能修改自身的计算机程序把自身也作为其加工的对象（数据）．

在计算机内，数据表现为一个记录，也称其为元素或节点，在图中又称之为顶点等．每个节点由计算机内存中的一个或多个相继的计算机字（或称之为内存单元，或简称为单元）组成，且每个节点又分成若干个字段（或称之为域）．最简单的情况是：一个节点仅由一个计算机字组成且该节点又只含一个字段．举例来说，可用相继的两个计算机字来表示一张扑克牌（这里每张扑克牌是一个节点），每张扑克牌的结构如下：

WORD i	TAG	SUIT	RANK	NEXT
WORD i+1	TITLE			

(1.1)

其中，WORD i 包含四个字段，WORD i+1 包含一个字段，且这些字段的定义如下：

TAG＝{1，当这张牌背向上时；0，当这张牌背向下时．

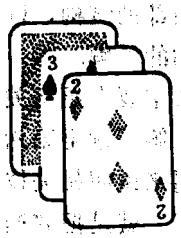
SUIT＝{1，牌花色等于梅花；2，牌花色等于方块；3，牌花色等于红心；4，牌花色等于黑桃．

RANK＝1，2，3，…，13，表示这张扑克牌的面值．

TITLE 是本张扑克牌助记的字符名称．

NEXT 是这叠纸牌中，紧接本张牌的下张牌（相应两个相继的计算机字）的首地址（即到下张牌的指针）．

书中，一个节点的地址，又称之为到（或指向）该节点的链接、指针或对该节点的访问等，即该节点的首地址．在上例中，域 NEXT 中存储的是下张牌的地址．按（1.1）的结构，我们可表示一叠扑克牌如下：



(a) 一叠纸牌

地址				
WORD 100	1	1	10	A
WORD 101	1	0	C	
WORD 386	0	4	3	100
WORD 387	3	S		
WORD 242	0	2	2	386
WORD 243	2	D		

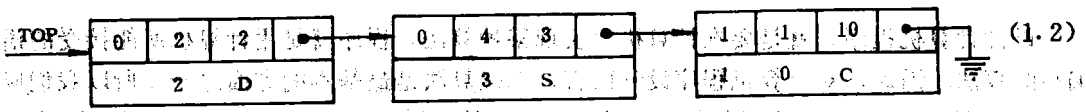
(b) 计算机表示

图 1.1 一叠纸牌的计算机表示

图 1.1 (a) 中，最底下的那张牌的地址是 100，中间那张牌的地址为 386，它的 NEXT＝

100 指出了它底下紧接它的牌是“最底下的”那张牌；最顶上那张牌的地址是 242，它的 NEXT =386 指出了它底下紧接它的牌是“中间”那张牌．最底下那张牌的 NEXT 值为 Λ，表示它的底下已没有牌，称 Λ 为空链接或接地．

链接是表示复杂数据结构关系的基本手段．通常我们可以用箭头来表达链接．上例中的链接关系可方便地表示如下：



(1.2)

这里，实际的地址 242，386 和 100 都已不再出现，因为就数据的逻辑关系而言，它们归根结底是非本质的．在（1.2）中，我们还加进了链接变量 TOP，也称为指针变量，即计算机程序内的一个变量，其值是一个指针或链接．现在，TOP 就指向顶上的那张牌．

要访问某节点内的诸子段，办法很简单，就是先给出要访问字段的名字，然后在其后的圆括号内写上指向该节点的链接．例如，按（1.1）和（1.2）有

TITLE(TOP)=2D

SUIT(TOP)=2

RANK(NEXT(TOP))=3

再如，由（1.1）和图 1.1，又有 RANK(100)=10.

有了访问节点和其中字段的办法，就能设计出基于（1.1）和（1.2）的各种运算（或操作）．

数据结构概念与数据概念不同，它一般包括如下三个方面：领域问题中数据间的逻辑关系，数据在计算机内的存储方式和施加于数据上的运算．

定义 1.1 一个数据结构就是按某一存储表示方式存储于计算机存储器中的，并由某组逻辑关系组织起来的一批数据和施加于其上的一组运算．

现在让我们一起来观察上面描述的一叠扑克牌：①叠中各张牌（节点）间的逻辑关系是一种简单的线性关系（由字段 NEXT 描述）；②这一叠牌（由（1.1）表示的对象），在计算机内存中采用链表方式存储，如图 1.1 或（1.2）所示；③由上面讨论的访问节点和节点字段的方法，可设计出这叠牌上的各种各样的运算（或曰操作，或曰算法），如在一叠扑克牌的顶上再放上一张面向上的新牌，计算叠中的牌数，等等．

综合上述，由①组织起来的一批数据（上述的一叠扑克牌），用②将其存储起来，并由③于其上定义了一个运算集合，就得到（或者说形成）了一个数据结构．

数据间关系的描述被称为数据的逻辑结构．数据的逻辑结构，可形式地表为一个二元组：L=（N，R），其中 N(L)是节点的有限集合，R(L)是 N 上的关系的集合．

设 L=（N，R）是一个逻辑结构，R=｛r｝（本书一般只讨论包含一个关系 r 的关系集合 R），若 a，b∈N，且（a，b）∈r，则称 a 是 b 的前驱节点，b 是 a 的后继节点，a 和 b 是相邻的节点．如果不存在 a∈N，使（a，b）∈r，则称 b 是始节点；如果不存在 a∈N，使（b，a）∈r，则称 b 是终节点；不是终节点和不是始节点的节点称为内节点．

数据的逻辑关系分为线性关系和非线性关系．线性关系的数据中有且仅有一个终节点和一个始节点，并且所有的节点都最多只有一个前驱和一个后继节点．非线性关系中最重要的一类是树形，树形中有且仅有一个没有前驱节点的节点，被称为根节点，其它节点仅有一个

前驱节点，并且对于非根节点都存在一条从根到该节点的路径（即关系 r 中的路径）. 非线性关系最一般情形是图，这种情况下，任意节点的前驱节点和后继节点的个数不受限制.

§3 算法的概念

设计计算机程序，就是要在计算机上实现某种算法，程序就是用计算机所能接受的语言编写的算法. 因此，对计算机程序设计而言，无疑算法是最基本的方面之一，所以我们应仔细讲解一下算法. 从某一个具体的且富有代表性的算法讲起，可能会更直观、更易于理解. 最著名算法之一是 Euclid（欧几里得）算法. 该算法就是在欧几里得的《几何原本》中所描述的求两个数的最大公因子的过程.

算法 E（m，n. n）

/ * 给定两个正整数 m 和 n，本算法求它们的最大公因数 * /

(1) / * 求余数 * /r←m− ⌊m/n⌋ * n.

/ * ⌊x⌋ 表示不超过 x 的最大整数；0≤r<n * /

(2) / * 余数为零？ * /IF r=0 THEN (PRINT n. RETURN).

/ * n 是答案 * /

(3) / * 互换 * /m←n. n←r.

GOTO (1) ■

在对算法 E 解释之前，我们先介绍一下算法描述语言，因为算法 E 是用 ADL 算法语言书写的. 常用的算法描述语言主要有类 PASCAL 语言，Knuth 在他的书（见文献〔9〕）中使用的语言（下面简称 Knuth 语言），等等. 类 PASCAL 语言过多涉及数据类型定义，使用不够简便. Knuth 语言由自然语言和数学语言组成，容易读，但用它书写递归算法不够方便. 为此，我们使用一种算法描述语言 ADL，其目的是用较直观和方便的方式勾画出算法的主要特征.

下面我们来介绍用算法描述语言 ADL 书写算法的约定和书写格式.

●算法都有一个名字，有相关的输入与输出，具体规定如下：

●算法〈标识符〉(变量 i_1，…，变量 i_m. 变量 j_1，…，变量 j_n)

其中〈标识符〉表示算法的名字.〈标识符〉是由字母和数字组成的有限符号串，且串中第一个符号必须是字母. 变量 i_k 为输入变量，m≥0. 当 m=0 时，表示没有输入变量. 变量 j_l 为输出变量，n≥0，通常，一个算法至少有一个输出；如果算法的输出特别明确时，往往省略输出变量.

●算法的每一步骤（或曰每组语句）都要有编号，编号为用圆括号括起来的正整数.

●算术表达式可使用常用的算术运算符，如+，−，*，/，DIV，MOD，⌊⌋（取地板运算），⌈⌉（取天棚运算，定义见附录），….

●逻辑表达式可使用关系运算符（=，≠，<，>，≤，≥），逻辑运算符（AND，OR，NOT）和逻辑常量（true，false).

●集合运算符∪，∩，−（差），⊂，⊃，….

●每条语句都用“.”作为结束符.

●赋值语句的形式如下：

a←b.

其中 a 是变量，b 是表达式. a↔b. 表示将变换变量 a 与 b 之内容进行交换. a←b←c. 表示将 c 的值同时赋给变量 a 和 b.

●条件语句有下述三种：

(1) IF〈逻辑表达式〉THEN
〈语句 1. …. 语句 m〉.

(2) IF〈逻辑表达式〉
THEN〈语句 1. …. 语句 m〉
ELSE〈语句 1. …. 语句 n〉. m，n≥1

(3) CASE DO
(〈逻辑表达式 1〉:〈语句 1. …. 语句 n_1〉.
⋮
〈逻辑表达式 m〉:〈语句 1. …. 语句 n_m〉).

其中 $n_i \geqslant 0$，$m \geqslant 1$，$1 \leqslant i \leqslant m$.

●循环语句有如下三种：

(1) WHILE〈逻辑表达式〉DO
〈语句 1. …. 语句 n〉.

其中 n≥1.

(2) FOR〈变量〉=〈算术表达式 1〉TO〈算术表达式 2〉STEP〈算术表达式 3〉DO
〈语句 1. …. 语句 n〉.

其中 n≥1. 若〈算术表达式 3〉≡1，则“STEP〈算术表达式 3〉”可略去.

(3) FOR ∀〈变量〉∈〈集合〉DO
〈语句 1. …. 语句 n〉.

其中 n≥1.

●转移语句如下：

GOTO (〈步骤号〉).

●以上各种语句，如果圆括号中的语句个数为 1 时，圆括号可以省略；否则一定要有.

●EXIT 语句可以在通常的结束条件满足之前，被用来结束 WHILE 或 FOR 循环的执行. EXIT 导致转移到紧接在包含 EXIT 的（最内层的）WHILE 或者 FOR 循环后面的一个语句.

●RETURN 语句是用来指出一个算法执行的终点；如果算法在最后一条指令之后结束，它通常被省略.

●每个算法都用■表示书写完毕. 注意算法不一定在■处运行结束.

●算法中的注释被括在括号/* */之中.

●诸如 READ 和 PRINT（打印输出信息）之类的各种输入或输出语句，在书写算法时也会用到.

上述有关 ADL 的介绍只为我们初步了解算法创造了条件，那么，有什么办法使我们能比较快捷和深入理解一个算法呢？一个好的或许是唯一的办法，就是通过实例进行试验，然后才能上升为理性认识，进而从理论上证明其正确性并分析其复杂性（见本书第二章）. 至此已到了深入理解算法 E 的时候了. 我们选取 m=119，n=544 以试验之，作辗转相除，如图 1.2 所示.

E	$\frac{m}{n}$ $\lfloor\frac{m}{n}\rfloor$ *n=q*n	m	n	r
		119	544	
(1)	$\frac{119}{544}$ 0			119
(2)				
(3)		544	119	
(1)	$\frac{544}{119}$ 476			68
(2)				
(3)		119	68	
(1)	$\frac{119}{68}$ 68			51
(2)				
(3)		68	51	
(1)	$\frac{68}{51}$ 51			17
(2)				
(3)		51	17	
(1)	$\frac{51}{17}$ 51			0
(2)		(119, 544) =17		

图 1.2 算法 E 运行的例子

其中 m=$\lfloor\frac{m}{n}\rfloor$ *n+r=q*n+r

总而言之，算法就是一个解决问题的办法，更确切地说，就是一组（有限个）规则（或曰语句），它们规定了解决特定问题的一系列操作. 算法有五大特性：

（1）有限性：算法必须在执行有限步后结束. 如算法 E，步骤（1）中有 r<n，故若 r≠0，则经过步骤（3）置 n←r 后，n 已减小. 所以对任给的初始值 n，步骤（1）中的 n 是逐次减小的，而正整数的严格递减序列必然终止于有限步.

（2）确定性：算法的每一步骤都是确定的，待执行的动作对每一情况都有严格确切的规定. 例如算法 E，在执行步骤（1）时，必须确切知道如何做 n 除 m 并求余数 r，并且必须保证 m 和 n 总是正整数. 事实上，在算法 E 初始时已假定是如此，在步骤（1）的下一次执行之前，必先执行步骤（3），即执行 m←n（n≠0），n←r（r≠0）. 由此可见步骤（1）中的 m，n 总是正整数.

（3）输入：算法有零个或多个输入. 这些输入取自确定的对象集合. 如算法 E 有两个输入 m 和 n，均取自正整数集合.

（4）输出：算法有一个或多个输出，即与输入有确定关系的量. 如算法 E 有一个输出，步骤（2）中的 n，是两个输入的最大公约数.

(5) 能行性：算法中有待执行之操作都十分基本，用手工也能在有限的时间内作完. 如算法 E 用到两正整数相除，判断一整数是否为零，赋值，整数间的乘法和减法，取地板，这些都是能行的. 非能行操作的例子有，要求计算不尽小数，要测试 "$x^n+y^n=z^n$ $(n\geqslant 2)$ 有无正整数解"，等等.

这就是算法的五大特性. 假若一组规则只有除有限性之外的其它四大特性，那么则应称其为计算方法. 即便是算法，从而必须具备有限性，但其有限性也有程度上的差异. 从计算机执行和实用而言，我们不仅要求算法结束于有限的步数，而且还要求很有限的步数，这就需要对其进行分析，确切说要进行时间复杂性分析（见第二章）.

前面我们对算法进行了描述并讨论了其五大特性，但都是非形式化的. 现在应给出算法的形式定义.

定义 1.2 一个计算方法就是一个自动机，即四元组 $\mathcal{A}=(S, X, Y, f)$，其中 S，X 和 Y 分别是计算的状态集合、输入集合、输出集合，X 和 Y 都是 S 的子集，而 f 是 S 的变换（计算规则），f 不变 Y 中的每个元素，即 $f(y)=y$，对 $\forall\, y\in Y$.

集合 X 中的每个输入 x 确定一个计算序列 $x_0, x_1, x_2, \cdots$ 如下：

$x_0=x$，而 $x_{k+1}=f(x_k)$，对于 $k\geqslant 0$.

这个计算序列说是终止于 k 步，如果 k 是使 $x_k\in Y$ 的最小正整数. 这时就说输入 x 产生输出 x_k（于是 $x_k=x_{k+1}=x_{k+2}=\cdots\in Y$）.

一个计算方法说是一个"算法"，如果对于 $\forall\, x\in X$，x 所确定的计算序列都终止于有限步.

【例 1.1】对算法 E，$X=\{(m, n)\mid m, n$ 是正整数$\}$，$Y=\{(n)\mid n$ 为正整数$\}$，$S=X\cup Y\cup\{(m, n, r, 1), (m, n, r, 2), (m, n, p, 3)\mid m, n, p$ 为正整数，r 为非负整数；1，2，3 分别表示步骤 (1)、步骤 (2) 和步骤 (3)$\}$.

f 定义为：

$f(m, n)=(m, n, 0, 1)$　　表示算法 E 以初始 m，n 进入 1

$f(m, n, r, 1)=(m, n, \frac{m}{n}$之余数$, 2)$　　执行 1 后进入 2

$$f(m, n, r, 2)=\begin{cases}(n) & \text{若 } r=0\\ (m, n, r, 3) & \text{若 } r\neq 0\end{cases}$$　　执行 2

实际上，执行步骤 (1) 后有

$$m=q*n+r,\ \text{整数 } q\geqslant 0,\ 0\leqslant r<n \tag{1.3}$$

若 $r=0$，则 $n\mid m$（n 整除 m），$n=\gcd(m, n)$，即步骤 (2) 的 n 是所求的最大公约数. 若 $r\neq 0$，由式 (1.3) 知

$$\begin{aligned}&m, n\ \text{之公约数}\mid r, n\\ &r, n\ \text{之公约数}\mid m, n\end{aligned}$$

亦即

$$\begin{aligned}&m, n\ \text{之公约数是}\ r, n\ \text{之公约数}\\ &r, n\ \text{之公约数是}\ m, n\ \text{之公约数}\end{aligned}$$

特别是

$$m, n\ \text{之最大公约数}=r, n\ \text{之最大公约数}$$

因此，步骤 (3) 不改变初始问题之答案. 于是乎，由步骤 (3) 返回步骤 (1)，经步骤 (1) 后又得到

$$m=qn+r,\ \text{整数 } q\geqslant 0,\ 0\leqslant r<n$$

如此经有限次循环后，必然得r=0，算法E最终于步骤(2)结束，得n=gcd(m, n)．总之，步骤(2)的n是两个输入的最大公约数．

f(n) = (n)　　2中的n是答案，若r=0

f(m, n, p, 3) = (n, p, p, 1)　　执行3 (m←n. n←p. GOTO 1)

前面给出的形式定义太广了，它已不具备“能行性”，因此在前面定义1.2中的“算法”二字加了引号，就是说定义1.2还不该说是算法的形式定义．定义1.2中的S可包括由手工无法计算的无穷序列，f也可能包含手工不能实施的操作．

为了达到“能行性”之要求，可对定义1.2中的自动机$\mathscr{A}$再施加种种限制．例如，安·安·马尔科夫算法（ТЕОРИЯ АЛГОРИФМОВ, А. А. МАРКОЪ, ИАН СССР, 1954）可定义如下：

$\mathscr{A}=(S_N, X, Y, f)$，N是非负整数，$S_N=\{(\alpha, j)\mid \alpha\in A^*, 0\leqslant j\leqslant N\}$，其中A是有限字符集合，$A^*$表示A上所有的串$\alpha=$“$\alpha_1\alpha_2\cdots\alpha_n$”（$n\geqslant 0$，诸$\alpha_i\in A$）的集合；$X=\{(\alpha, 0)\mid \alpha\in A^*\}$，$Y=\{(\alpha, N)\mid \alpha\in A^*\}$．通过两组串$\theta_i$，$\varphi_i$和两组非负整数$j'$，$j''$（$0\leqslant j', j''\leqslant N$），其中$j=0, 1, \cdots, N-1$；定义f如下：对于$j=0, 1, 2, \cdots, N-1$，

$$f(\alpha, j)=\begin{cases}(\alpha, j') & \text{若 } \theta_j \text{ 不出现于 } \alpha \\ (\alpha\,[\theta_j\leftarrow\varphi_j], j'') & \text{若 } \theta_j \text{ 出现于 } \alpha\end{cases}$$

执行步骤j后，转步骤j'，

执行步骤j后，转步骤j''

（其中$\alpha\,[\theta_j\leftarrow\varphi_j]$表示以$\varphi_j$去替换$\theta_j$在$\alpha$中的最早出现），

而 $f(\alpha, N)=(\alpha, N)$　　输出(α, N)，结束于步骤N．

如此定义的计算方法是能行的，因为这种替换能用手工实行，而S_N中的每个元素也能手工计算．经验证明这足够广了，已经包括了一般的能行算法．

此外，还可用Turing机来形式化能行的计算方法．

习题一

1. 树形定义为：(a) 有一个称为根的节点没有前驱节点；(b) 除根外的其余节点仅有一个前驱节点；(c) 对于非根节点都存在一条从根到该节点的路径. 试问，如果从定义中去掉第(c)条，这时所定义的还是树形吗？并说明理由.

2. 请回答(1.2)链表中，SUIT(NEXT(TOP))=?，NEXT(NEXT(NEXT(TOP)))=?.

3. 给出一个算法，它在一叠牌（可能空，即 TOP 可能等于 Λ）的底下放进一张背向上的新牌（假定指针 newcard 指向这张新牌）.

4. 设一叠扑克牌的表示如(1.2)所示. 请给出一个算法，该算法把每张牌翻转过来.