数据结构

朱保平　俞研 ◎ 编著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

责任编辑：江　立
封面设计：ZTSA 中通世奥图文设计 专家

数据结构

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

通信地址：北京市海淀区中关村南大街5号
邮政编码：100081
电　　话：010-68948723　82562903
网　　址：www.bitpress.com.cn

关注理工高教
获取优质学习资源

ISBN 978-7-5682-9910-7
9 787568 299107 >
定价：58.00元

数据结构

朱保平　俞　研　编著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

内容简介

本书借鉴国内外高等院校《数据结构》相关教材，详细介绍了数据结构的基本理论和基本算法，内容包括绪论、线性表、栈和队列、串、数组和广义表、树和二叉树、图、查找、内部排序、外部排序。

本书内容丰富，案例翔实，既注重理论知识描述，又强调工程应用和复杂问题求解，可作为高等院校计算机科学与技术及相关专业“数据结构”课程教材，也可作为教师、研究生或软件技术人员的参考用书。

图书在版编目（CIP）数据

数据结构 / 朱保平，俞研编著. —北京：北京理工大学出版社，2021.6

ISBN 978-7-5682-9910-7

Ⅰ. ①数… Ⅱ. ①朱… ②俞… Ⅲ. ①数据结构-高等学校-教材 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字（2021）第 108336 号

出版发行 / 北京理工大学出版社有限责任公司

社　　址 / 北京市海淀区中关村南大街 5 号

邮　　编 / 100081

电　　话 / （010）68914775（总编室）

（010）82562903（教材售后服务热线）

（010）68944723（其他图书服务热线）

网　　址 / http：//www.bitpress.com.cn

经　　销 / 全国各地新华书店

印　　刷 / 唐山富达印务有限公司

开　　本 / 787 毫米×1092 毫米　1/16

印　　张 / 19

字　　数 / 443 千字

版　　次 / 2021 年 6 月第 1 版　2021 年 6 月第 1 次印刷

定　　价 / 58.00 元

责任编辑 / 江　立

责任校对 / 周瑞红

责任印制 / 李志强

前　言

计算机发展初期主要集中于数值计算，软件设计者将主要精力用于程序的设计，并不需要花太多的时间和精力在数据的组织上。

随着计算机应用领域的扩大、信息量的增加、信息范围的拓宽，非数值计算问题占据了当今计算机应用的大多数领域，简单的数据类型已不能满足现代计算机技术的需要，计算机系统程序和应用程序均会使用各种复杂的数据关系。因此，掌握“数据结构”课程知识能够让软件设计者正确选择和使用数据关系，对客观世界中的相关数据进行存储，构建复杂问题的解决方案。

在计算机科学中，“数据结构”不仅是一般非数值计算程序设计的基础，而且是“数据库原理”“编译原理”“操作系统”“软件工程”和“人工智能”等课程的基础。数据结构技术也广泛应用于信息科学、系统工程等工程技术领域。

本书强调抽象问题描述和复杂问题求解等内容，由作者结合“数据结构”课程长期教学经验编著而成。教材以“数据结构”课程的重要知识点为纽带，构建复杂问题的解决方案，夯实程序设计基础，拓展数据和关系的表示方法，强化从实例计算到模型计算方法的思路，帮助读者提高利用专业知识解决复杂问题的能力。

全书内容丰富，案例翔实，既注重理论知识描述，又强调工程应用和复杂问题求解，主要包括绪论、线性表、栈和队列、串、数组和广义表、树和二叉树、图、查找、内部排序和外部排序 10 章。

本书第 1 章 ~ 第 8 章由朱保平编著，第 9 章 ~ 第 10 章由俞研编著，张宏教授对本书内容提出了宝贵的意见。

由于水平有限，书中难免存在疏漏和不足之处，恳请读者批评指正。

作者于南京理工大学

2021. 2

目 录

第1章 绪 论

一个好的程序需要选择合理的数据结构和算法，而算法的选择取决于描述实际问题的数据结构。因此，为了编写出一个好的程序，必须分析待处理数据的特征、数据间的相互关系以及数据在计算机内的存储表示，并利用这些特性和关系设计出相应的算法与程序。

1.1 数据结构概述

“数据结构”是一门介于数学、计算机硬件和计算机软件之间的计算机科学领域的核心课程。瑞士计算机科学家尼古拉斯·沃斯(Niklaus Wirth)教授指出，“算法+数据结构=程序”。算法是解决特定问题的步骤和方法，数据结构是问题的数学模型，程序则是为计算机处理问题编制的一组指令集。计算机的算法与数据结构密切相关，算法依赖具体的数据结构，数据结构直接影响算法的选择和效率。

数据的运算是定义在数据的逻辑结构之上的，每种逻辑结构有一组相应的运算。非数值计算问题的常见运算有查找、插入、删除、更新和排序等。也就是说，数据结构需要给出每种结构类型所定义的各种运算的算法。

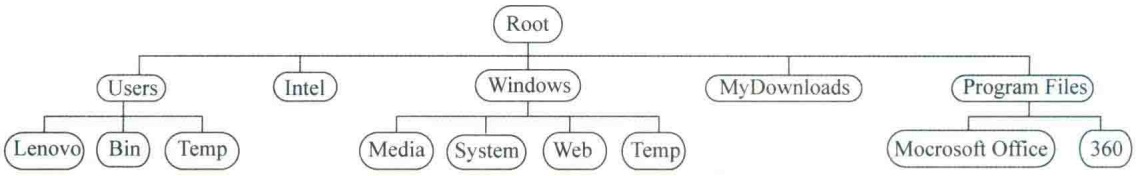
典型的数据结构有表、树和图，如图1.1所示。

车次	发站—到站	出发时间	到达时间	运行时间
G204	南京南—北京南	7:16	12:13	4 h 57 min
G102	南京南—北京南	7:48	12:29	4 h 41 min
G104	南京南—北京南	8:02	12:33	4 h 31 min
G6	南京南—北京南	8:13	11:36	3 h 23 min
G34	南京南—北京南	8:33	13:07	4 h 34 min

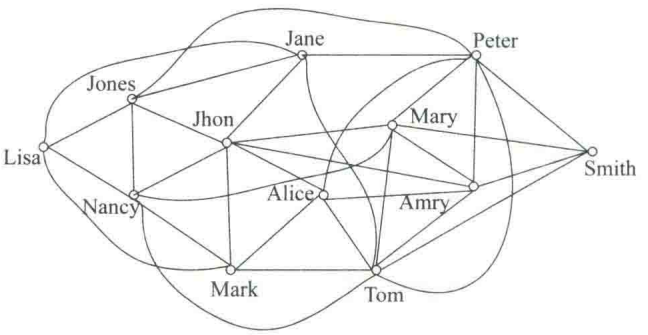
(a)

图1.1 典型的数据结构

(a)列车时刻信息表



(b)



(c)

图 1.1　典型的数据结构(续)

(b) Windows 文件系统的目录结构；(c) 社交网络拓扑结构

Windows 文件系统的目录是一种典型的树形结构，由大量网络结点组成的社交网络拓扑结构是典型的图状结构。

从以上 3 个非数值计算的例子可以看出，描述这类问题的数学模型不再是数学方程，而是表、树和图之类的数据结构。因此，概括地说，数据结构是研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作的一门科学。

1.1.1　数据结构的研究对象

计算机程序对数据进行加工处理，一般情况下，这些数据之间存在着一定的关系。当计算机程序所涉及的运算对象是数值型数据时，软件设计者的主要精力用于程序的设计，而不需要花太多的时间和精力在数据的组织上；当计算机处理非数值计算问题时，所涉及数据之间的关系可能非常复杂，很多问题甚至无法用数学方程式描述。数据结构在非数值计算中显得尤为重要。

下面分别从数值计算和非数值计算的角度，给出两个典型的实例。

1. 数值计算问题

阿克曼(Ackermann)函数是非原始递归函数，它需要两个自然数作为输入值，输出一个自然数。Ackermann 函数的数学公式如下：

$$ack(m,\ n)=\begin{cases}n+1 & (m=0,\ n\geqslant 0)\\ ack(m-1,\ 1) & (m>0,\ n=0)\\ ack(m-1,\ ack(m,\ n-1)) & (m>0,\ n>0)\end{cases}$$

用递归方法求解 ack(m, n)的递归程序如下：

```
int ack(int m,int n)
  if(m==0&&n>=0)return n+1;
```

```
    else if(m>0&&n==0)return ack(m-1,1);
        else if(m>0&&n>0)return ack(m-1,ack(m,n-1));
            else exit(-1);                    //输入数据不合法,异常处理
}
```

Ackermann 函数递归调用的次数增长非常快，采用递归方法只能求解 m 和 n 很小的值。显然，局限于递归算法，用手工求解这个问题是极其困难的，甚至是不可能的。要想用手工求解这个困难问题，软件设计者需要改进程序的设计技巧。

2. 非数值计算问题

“井”字游戏是一种在 3×3 格子上进行的连珠游戏，两个游戏者轮流在格子里留下标记（一般先手者为×，另一个人为○），最先在任意一条直线上成功连接 3 个标记的一方获胜。图 1.2 给出了“井”字游戏的一个格局及部分博弈树。在博弈过程中，两个选手都遵循最优策略。最优策略是一组用来说明一个选手如何移动来赢得游戏的规则。第一个选手的最优策略是把自己的得分最大化的策略，第二个选手的最优策略是把对手得分最小化的策略。

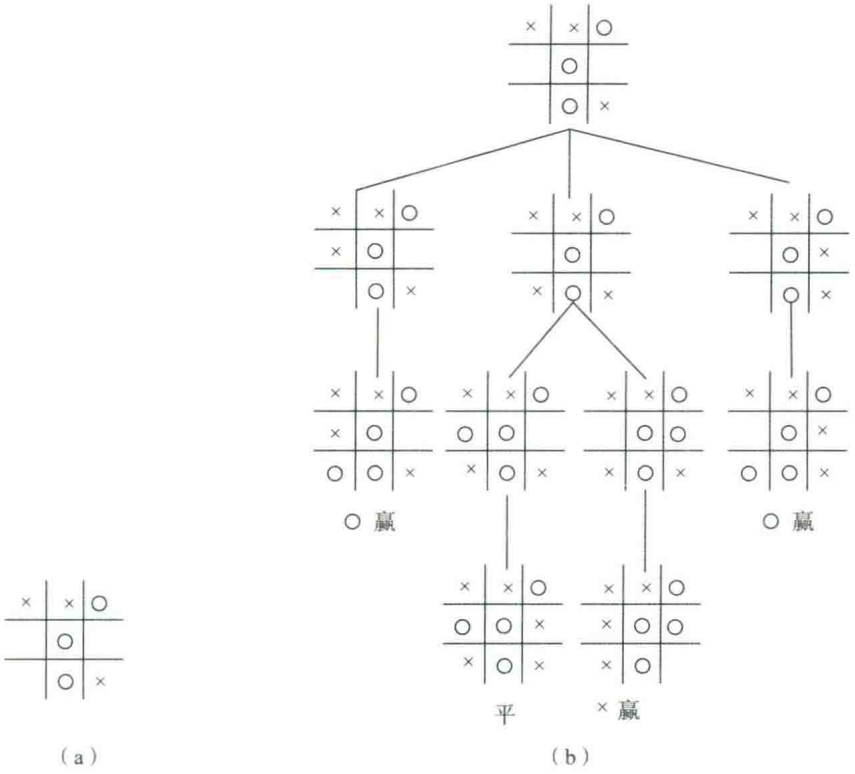


图 1.2 “井”字游戏格局间的关系

(a)“井”字游戏的一个格局；(b)“井”字游戏的部分博弈树

1.2 数据结构的相关概念

数据是信息的符号表示，在计算机科学中指所有能输入到计算机中并被计算机程序处理

的符号的总称。从计算机科学的角度讲，数据的含义极为广泛，如整数、实数等为数值型数据，文字、图形、图像和声音等多媒体信息为非数值型数据。数值型数据和非数值型数据都可以通过编码成为计算机可以识别的数据。

数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一般来说，能独立、完整地描述问题的一切实体都是数据元素。在“井”字游戏中，一个格局就是一个数据元素，对于此类非数值计算问题，除了描述数据元素以外，还需要重点描述数据元素之间的逻辑关系。一个数据元素可由若干个数据项组成。

数据项是数据的不可分割的最小单位。例如，高铁列车时刻信息表中一个车次的信息为一个数据元素，而车次信息中的每一项(如车次、发站—到站、出发时间、到达时间和运行时间)为一个数据项。

数据对象是性质相同的数据元素的集合，是数据的一个子集。在实际应用中处理的数据元素通常具有相同的性质，例如，高铁列车时刻信息表中每个数据元素具有相同数量和类型的数据项，所有数据元素(车次信息)的集合构成一个数据对象。

数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。数据元素不是孤立存在的，它们之间存在着某种关系，这种数据元素之间的关系称为**结构**。数据结构由数据元素的集合和数据元素之间关系的集合组成，可以表示为二元组：

$$\text{Data\_Structure}=(D, R)$$

其中，D 是数据元素的有限集，R 是 D 上二元关系的集合。

根据数据元素之间关系的不同特性，数据结构通常可以分为集合结构、线性结构、树形结构和图状结构，如图 1.3 所示。

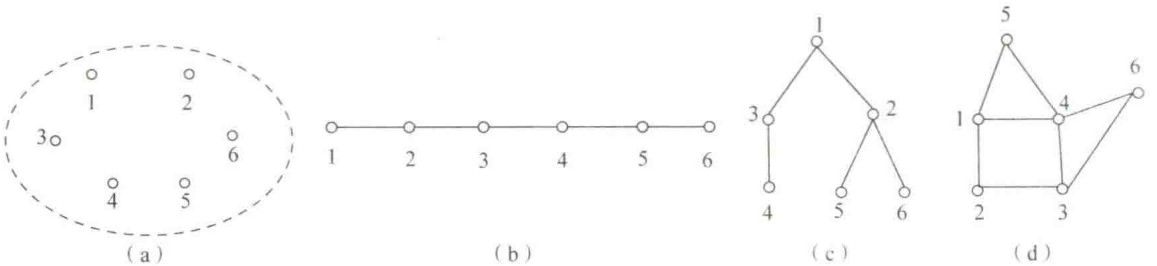


图 1.3　4 种基本的数据结构

(a)集合结构；(b)线性结构；(c)树形结构；(d)图状结构

(1)**集合结构**：数据元素属于同一个集合，除此之外没有任何关系，即元素之间是空关系。例如，$D=\{1, 2, 3, 4, 5, 6\}$，$R=\varnothing$，如图 1.3(a)所示。集合结构是一种松散的数据结构，在实际问题中，往往需要借助其他结构来表示。

(2)**线性结构**：数据元素之间是一种线性关系，即元素之间存在着一对一的关系。例如，$D=\{1, 2, 3, 4, 5, 6\}$，$R=\{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)\}$，如图 1.3(b)所示。线性结构中开始结点和终端结点是唯一的，其余结点有且仅有一个前驱和一个后继。

(3)**树形结构**：数据元素之间是一种层次关系，即元素之间存在着一对多的关系。例

如，D={1，2，3，4，5，6}，R={(1，2)，(1，3)，(3，4)，(2，5)，(2，6)}，如图1.3(c)所示。树形结构的每个结点最多只有一个前驱结点，但可以有多个后继结点，且终端结点可以有多个。

(4)**图状结构(也称网状结构)**：数据元素之间是一种任意关系，即元素之间存在着多对多的关系。例如，D={1，2，3，4，5，6}，R={(1，2)，(1，5)，(2，3)，(3，4)，(4，1)，(4，5)，(4，6)，(3，6)}，如图1.3(d)所示。图状结构每个结点的前驱结点和后继结点的数量可以是任意的，可能没有开始结点和终端结点，也可能有多个开始结点和终端结点。

树形结构和图状结构统称为非线性结构，其结点之间存在一对多或多对多的关系。线性结构是树形结构的特例，而树形结构是图状结构的特例。

数据结构有逻辑结构和物理结构两个层次。

逻辑结构是数据元素之间逻辑关系的整体，逻辑关系是数据元素之间的关联方式或邻接关系，与数据自身的存储无关。它是从具体问题抽象出来的数学模型，用来描述数据元素及其关系的数学特性。

物理结构(也称存储结构)是数据结构在计算机中的表示或映像，研究数据结构在计算机中的表示方法，包括数据结构中数据元素的表示和数据元素之间关系的表示。

数据的存储结构除了存储数据元素之外，必须隐式或显式地存储数据元素之间的逻辑关系。通常数据元素在计算机中有顺序映像和非顺序映像两种不同的表示方法，由此得到顺序存储结构和链式存储结构两种不同的存储结构。

顺序映像借助数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系，用一组地址连续的空间存放数据元素，逻辑上相邻的数据元素物理上一定相邻，由此得到的存储表示称为**顺序存储结构**。顺序存储结构是基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

非顺序映像借助指示数据元素存储地址的指针表示数据元素之间的逻辑关系，用一组地址任意(可以连续也可以不连续)的空间存放数据元素，逻辑上相邻的数据元素物理上不一定相邻，数据元素之间的逻辑关系通过指针来表示，由此得到的存储表示称为**链式存储结构**。链式存储结构也是基本的存储表示方法，通常借助于程序设计语言中的指针来实现。

例如，线性表(2，4，6，8，10)的存储结构如图1.4所示。

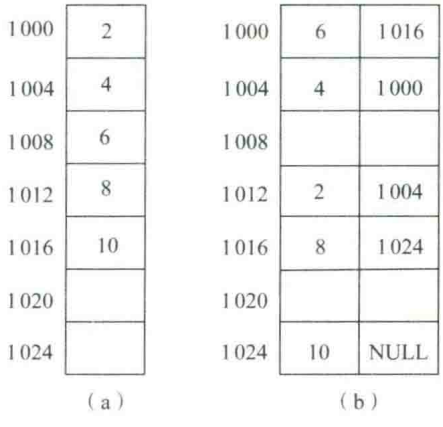


图1.4 线性表的存储结构

(a)顺序存储；(b)链式存储

1.3 数据类型和抽象数据类型

在程序设计中，数据和运算是两个不可或缺的因素。最初的机器语言和汇编语言中的数据没有数据类型的概念。

数据类型是一组值的集合以及定义在该值集上的一组操作的总称。数据类型和数据结构密切相关，它最早出现在高级程序语言中，用来描述程序中操作对象的特性。在用高级程序语言编写的程序中，每个变量、常量或表达式都有确定的数据类型。类型显式或隐式地规定了在程序执行期间变量或表达式所有可能取值的范围，以及在此之上允许进行的操作。例如，整型数据类型是指一组值的集合 Z={…，-3，-2，-1，0，1，2，3，…}和定义在该值集上的加、减、乘、除和模运算等算术运算的一组操作。

抽象数据类型(Abstract Data Type，ADT)是用户进行软件设计时从问题的数学模型中抽象出来的逻辑数据结构以及该逻辑数据结构上的一组操作。抽象关注问题的本质特征而忽略非本质的细节，是对具体事务的概括。抽象数据类型中的数据对象、数据操作的声明与数据对象的表示、实现相互分离。

抽象数据类型有两个重要特征：数据抽象和数据封装。数据抽象用 ADT 描述程序处理的实体时，强调数据的本质特征、所能完成的功能及其与外部用户的接口(外界使用数据的方法)。数据封装将实体的外部特性与其内部的实现细节分离，并且对外部用户隐藏其内部实现细节。

抽象数据类型一般由数据元素、数据关系和基本操作组成，可以表示为三元组：

$$ADT=(D，R，P)$$

其中，D 表示数据对象，R 表示 D 上的数据关系集合，P 表示 D 中数据对象的基本操作集合。

抽象数据类型定义的基本格式如下：

```
ADT 抽象数据类型名|
    数据对象:数据对象的定义
    数据关系:数据关系的定义
    基本操作:基本操作的定义
|ADT 抽象数据类型名
```

其中，基本操作的定义格式如下：

```
基本操作名(形式参数表):操作功能描述
```

例 1.1 定义复数的抽象数据类型，对复数进行构造和销毁，并返回复数的实部与虚部，以及两个复数之和。

【解】复数的抽象数据类型定义如下：

```
ADT Complex|
数据对象:
    D=|a,b|a,b 均为实数|
数据关系:
    R=|(a,b)|a 为复数的实数部分,b 为复数的虚数部分|
基本操作:
    AssignComplex(&z,x,y):构造复数 z,x 为实部,y 为虚部,& 表示返回操作结果
    DestroyComplex(&z):销毁复数 z
    GetReal(z,&real):用 real 返回 z 的实部
```

GetImag(z,&imag)：用 imag 返回 z 的虚部

Add(z_1,z_2,&sum)：用 sum 返回复数 z_1 和 z_2 的和

ADT Complex

例 1.2 定义 n 元组的抽象数据类型，实现 n 元组的构造、销毁，返回和更新指定位置的值，并对 n 元组进行排序。

【解】n 元组的抽象数据类型定义如下：

ADT N_tuple

数据对象：

$D=\{a_1,a_2,a_i,\cdots,a_n \mid a_i$ 均为整型数$\}$

数据关系：

$R=\{(a_{i_1},a_{i_2},a_{i_j},\cdots,a_{i_n}) \mid i_j \in \{1,2,\cdots,n\}\}$

基本操作：

InitN_tuple(&t,v_1,v_2,$\cdots$,v_n)：构造 n 元组

DestroyN_tuple(&t)：销毁 n 元组

Geti(t,i,&e)：用 e 返回第 i 元的值

UpData(&t,i,e)：用 e 更新第 i 元的值

SortN_tuple(&t)：对 n 元组排序

ADT N_tuple

抽象数据类型可通过基本数据类型来表示和实现，即利用处理器中已经存在的数据类型说明新的结构类型，用已经实现的操作组合实现新的操作。本书采用 C++中的类作为抽象数据类型的描述工具。

1.4 算法及其描述

算法和数据结构密切相关，算法设计前要先确定相应的数据结构，而在讨论某种数据结构时，也必然要设计相应的算法。

1.4.1 算法的特性

算法是对特定问题求解步骤的一种描述，是指令的有限序列，每条指令表示计算机的一个或多个操作。

例 1.3 设计一个算法，计算 $z=[\sqrt{x}]$（[]表示取整）。

【解】该算法的步骤为：

(1)赋初值(a, b, c)=(0, 0, 1)；

(2)计算 b=b+c；

(3)如果 b>x，转(6)；

(4)计算(a, c)=(a+1, c+2)；

(5)转(2)；

(6)赋值 z=a；

(7)输出 z;

(8)算法结束。

算法具有以下 5 个特性:

(1)**有穷性**:对于任何一个合法的输入值,一个算法必须总是在执行有穷步后结束,且每一步都在有限的时间内完成。

(2)**确定性**:算法中每一条指令必须有确切的含义,不存在二义性,即在任何条件下,算法只有唯一的一条执行路径。

(3)**可行性**:算法可以通过已经实现的基本运算执行有限次来实现。

(4)**输入**:算法的输入取自于某个特定对象的集合,作为算法的加工对象,通常为算法中的一组变量。有些输入量需要在算法执行过程中输入,有些算法表面上没有输入,实际输入量已被嵌入在算法之中。

(5)**输出**:一个算法有一个或多个输出,这些输出是一组与输入有着某些特定关系的量,是算法进行信息处理后的结果。

算法代表了对问题的求解步骤,程序是算法在计算机上使用某种程序设计语言的具体实现。原则上,算法可以用任何一种程序设计语言实现。两者的区别是,算法必须满足有穷性,而程序可以不满足有穷性。例如,Windows 操作系统在用户未操作时一直处于"等待"的循环中,此时程序是无限循环的,直到用户进行操作为止。

1.4.2 算法描述

算法可以采用自然语言方式描述,如例 1.3 中的算法描述,也可以采用图形方式(如流程图、拓扑图等)描述,如图 1.5 所示。如果用程序设计语言描述算法,算法则表现为程序。用程序设计语言描述例 1.3 的算法如下:

```
void squareroot(int x){
  int a=0,b=0,c=1;
  b=b+c;
  while(b<=x){
    a=a+1;
    c=c+2;
    b=b+c;
  }
  z=a;
  cout<<z;
}
```

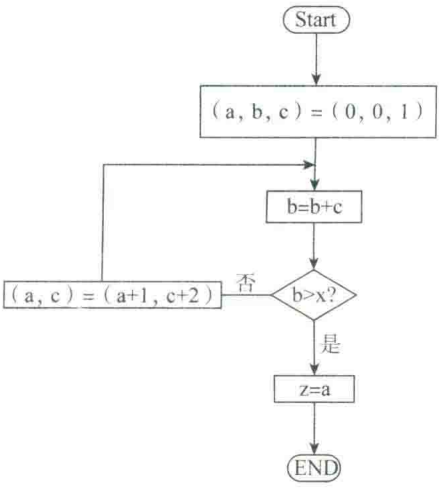


图 1.5 例 1.3 算法的流程图

1.4.3 算法设计的要求

同一个问题可能有多种求解的算法,一般来说,一个好的算法应该满足以下几个要求。

(1)**正确性**：算法能够正确执行，满足当前具体问题的需求，即算法的执行结果能够满足预定的功能和性能需求。这是算法最重要也是最基本的要求。

一个算法满足正确性一般分为4个层次：

①没有语法错误；

②随意输入几组数据能够得出符合要求的结果；

③精心设计的、典型的、苛刻的合法输入能够得出符合要求的结果；

④所有合法的输入数据能够得出符合要求的结果。

(2)**可读性**：算法应当可读，有利于阅读者理解程序。算法主要是为了人的阅读与交流。为了达到这一要求，算法的逻辑必须是清晰的、简单的和结构化的。在算法中必须加入注释，简要说明算法的功能、输入与输出参数的使用规则、重要数据的作用和算法中各程序段完成的功能等。

(3)**健壮性**：算法应具有容错性和例外处理能力。正确的输入应该有正确的输出。对于错误的输入，算法应该给出适当的反应，不会产生错误动作或陷入瘫痪。

(4)**高效率与低存储量需求**：效率是算法执行所需的时间，算法的存储量是算法执行过程中所需要的最大存储空间。效率、存储量一般与问题的规模有关。

1.5 算法分析

完成一个算法设计后，需要对算法进行分析，确定算法的优劣。通常为了满足算法设计的要求，需要进行算法效率分析和算法存储空间分析等。

1.5.1 算法效率的分析方法

通常有两种分析算法效率的方法：事后统计方法和事前分析估算方法。

1)事后统计方法

事后统计方法收集算法的执行时间和实际占用空间的统计资料。这种方法有两个缺陷：一是必须先运行依据算法编写的程序；二是存在其他因素掩盖算法的本质，如计算机的硬件、软件等环境因素。

2)事前分析估算方法

一个算法转换成高级语言编写的程序在计算机上实现时，所耗费的时间与以下因素有关：①计算机的运行速度；②编写程序采用的计算机语言；③编译产生的机器语言代码质量；④问题的规模。

1.5.2 时间复杂度

不考虑计算机硬件和软件有关的因素，仅考虑算法本身效率的高低时，算法的效率只依赖问题的规模(用整数n表示)，是问题规模的函数。

事前分析估算方法通过分析问题的规模，求出算法的时间界限函数。时间界限函数一般可以表示为

$$f(n)=a_m n^m+a_{m-1}n^{m-1}+\cdots+a_2 n^2+a_1 n+a_0$$

一个算法通常由控制结构(顺序、选择和循环3种结构)和操作构成，算法的运行时间由两者的综合影响。为了比较同一问题的不同算法，通常从算法中选取对于所研究的问题来说是基本操作的原操作，以该原操作重复执行的次数(也称语句频度)度量算法的时间。一般情况下，算法中原操作重复执行的次数是问题规模n的函数f(n)，算法的时间度量记作：

$$T(n)=O(f(n))$$

它表示随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同，称作算法的渐近时间复杂度，简称时间复杂度。

例1.4 计算1+2+3+…+n的算法如下，分析其时间复杂度。

```
int sumn(int n){
  int sum=0;
  for(int i=1;i<=n;i++)
      sum+=i;
  return sum;
}
```

【解】该算法的基本运算是sum+=i，其语句频度为

$$T(n)=n=O(n)$$

该程序段的时间复杂度为O(n)，通常称为线性阶。

例1.5 已知n为正整数，计算$2^0+2^1+2^2+\cdots 2^i+\cdots$，($2^i<n$)的算法如下，分析其时间复杂度。

```
int sumpower(int n){
  int sum=0,i=1;
  while(i<n){
    sum+=i;
    i=i*2;
  }
  return sum;
}
```

【解】该算法的基本运算是sum+=i，由题意知，while循环中i的值依次为1，2，4，8，16…，其语句频度为

$$T(n)=\log_2 n=O(\log_2 n)$$

该程序段的时间复杂度为$O(\log_2 n)$，通常称为对数阶。

例1.6 已知选择排序算法如下，分析其时间复杂度。

```
void selectsort(int a[],int n){
  int i,j,k,temp;
  for(i=0;i<n;i++){
    k=i;
    for(j=i+1;j<n;j++)
```