程序员级高级程序员级程序设计

张福炎 主编
郑国梁 主审

中国计算机软件专业技术资格和水平考试
统编辅导教材

清华大学出版社

中国计算机软件专业技术资格
和水平考试统编辅导教材

程序员级高级程序员级
程序设计

张福炎　　主编
郑国梁　　主审

清华大学出版社

内 容 简 介

本书是中国计算机软件专业技术资格和水平考试委员会组织编写的全国统编辅导教材之一，它依照1991年4月公布的《中国计算机软件专业技术资格和水平考试暂行规定与考试大纲》进行编写。内容包括：程序设计方法与常用算法；基本数据结构及操作；动态数据结构及操作；排序与查找；文件与文件I/O；事务处理流程图；以及CASL程序设计。全书除简明扼要地叙述了一些概念和基本算法原理之外，还编写了许多例题和习题，以指导软件人员参加下午题的考试。

本书可供参加资格和水平考试的计算机应用软件人员学习、参考，也可作为大专院校有关专业的师生及广大计算机工作者的常用参考书。

（京）新登字158号

TP316/37 09

程序员级高级程序员级程序设计

张福炎　主编

郑国梁　主审

☆

清华大学出版社出版

北京　清华园

北京顺义曙光印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16　印张：27.125　字数：693千字

1992年8月第1版　1992年8月第1次印刷

印数：00001—20000

ISBN 7-302-01109-5/TP·417

定价：15.50 元

出 版 说 明

当今国际间的竞争是综合国力的竞争，关键又是科学技术的竞争，说到底是人才的竞争。而电子信息技术又是当今国际间竞争的热点，它渗透和影响着现代化社会生活的各个方面。而科学技术（包括电子信息技术）是要靠人去掌握、去应用、去发展的，国家的强盛、民族的振兴靠人才，人才的培养靠教育。所以，党中央要求我们把经济建设转到依靠科技进步和提高劳动者素质的轨道上来。

培养人才要坚持多种形式、多种途径，大力开展岗位培训，不断提高职工队伍的技术和专业水平。计算机软件专业技术资格和水平考试制度，就是为了加速我国电子信息技术的广泛应用和软件事业的发展，科学考核和合理使用人才，促进计算机软件人才的国际交流与合作，进一步深化职称改革。这种考试是于1985年首先在上海、云南、四川三省市实行的，1987年发展为部分省、区、市的联合考试，到1988年全国已经有31个省、区、直辖市和计划单列市参加程序员、高级程序员两个级别的联合考试，1989年发展为程序员级、高级程序员级、系统分析员级三个级别的联合考试，1990年在全国统一组织实施了软件专业技术职务任职资格（水平）考试。1991年3月计算机软件专业技术资格和水平考试工作会议，对考试《暂行规定》作了修改，新的《暂行规定》把“资格”考试与“水平”考试从性质上和考试级别上作了区分，“资格”考试除了保留“程序员”、“高级程序员”的级别外，增加了“初级程序员”的考试级别，而“水平”考试则仍为“程序员”、“高级程序员”、“系统分析员”三个级别。同时在报考条件与考试的内容和评分标准等方面，也都作了相应的规定。1991年的软件专业技术资格和水平考试就是依照新的《暂行规定》进行的。实践证明这是一种严格的认定考试制度，给应试者提供了一次均等的机会。软件专业技术资格和水平考试，每年举行一次，实行全国统一组织、统一大纲、统一试题、统一评分标准。

中国计算机软件专业技术资格和水平考试委员会（以下简称考委会）把考试和培训两项相辅相成的工作担当起来是责无旁贷的。为了使全国参加统一考试的考生得到较好的辅导，编一套全国统一的辅导教材是完全必要的。1991年9月考委会在北京主持召开了辅导教材编审委员会会议，对统编辅导教材的编写出版作了部署，成立了统编辅导教材编审委员会，并确定了这套教材六本书的主编和主审，他们是：《初级程序员级软硬件知识》（主编：刘尊全；主审：吴克忠）、《程序员级高级程序员级硬件知识》（主编：王爱英；主审：周明德）、《程序员级高级程序员级软件知识》（主编：施伯乐；主审：潘锦平）、《计算机综合应用知识》（主编：罗晓沛；主审：侯炳辉）、《程序员级高级程序员级程序设计》（主编：张福炎、钱士钧；主审：郑国梁）、《1990年度—1991年度试题分析与解答（初级程序员级、程序员级、高级程序员级、系统分析员级）》（主编：张然；主审：施伯乐）。我们的本意是要把这套全国统编辅导教材搞成为具有正确性、科学性、系统性，而且针对性强，在一段时间内相对稳定的好教材。虽然吸取了北京、上海、成都等地已有的辅导教材的长处，但由于时间紧，经验不足，错误在所难免，请读者和有关专家能给予指正。

教材编审委员会

秘书组联络地址：北京市海淀区学院南路55号（邮政编码 100081）

前　言

计算机软件专业技术资格和水平考试是当前深化职称改革、完善专业技术职务聘任制的一项重要措施，也是造就宏大计算机软件人才队伍的一个有力手段。为了指导广大软件人员的学习，帮助准备参加软件专业技术资格（水平）考试的学员进行复习应试，中国计算机软件专业技术资格和水平考试委员会组织编写了这套辅导教材。本书是适用于程序员级和高级程序员级的程序设计辅导教材，它以1991年4月公布的《中国计算机软件专业技术资格和水平考试暂行规定与考试大纲》和统编教材编审委员会拟定的教材大纲为依据，针对程序员级、高级程序员级考试下午试题中对于程序分析、设计和编码等方面的要求而编写的。

考虑到参加软件专业技术资格(水平)考试的广大学员一般都已熟悉一门或二门程序设计语言，并已具有基本的编程能力。因此，本书的目的是希望对读者原有的程序设计知识和编程能力起到总结、拓宽和提高的作用，增加程序的设计和阅读能力，并掌握各类常用算法的原理与技术要点。

全书共分七章。第一章概括介绍程序设计方法和常用的数值计算和非数值计算算法。第二章系统地介绍数组、队列、栈、集合和串等程序设计中常用的基本数据结构。第三章进一步阐述各种动态数据结构，如链表、树和图以及它们的各种操作。第四章对各种常用排序算法和查找算法进行了对比和分析。第五章在扼要叙述文件的基本概念后，重点对 FORTRAN、C、PASCAL 和 COBOL 语言中的文件I/O操作进行了概括比较。第六章通过实例对事务处理及流程图设计进行了阐述。第七章以大量例题详细介绍了 CASL 汇编语言及其程序设计的方法和技巧。

在编写过程中，编者力求使本书内容符合考试大纲的规定和要求，注意做到基本概念和算法原理的叙述简明、准确。书中包含了大量的例题和习题，它们均用C语言写成（第五、第七章除外），并已在机器上进行了调试和验证。

本书由张福炎、钱士钧主编，郑国梁主审。参加编写工作的有钱士钧（第一、六章）、陈道蓄（第二章）、陈本林（第四章）、叶晓峰（第三、五章）和管有庆（第七章）。孔宪礼同志在全书编写过程中做了大量工作，统编教材编委会秘书组为本书的编写给与了许多帮助，在此一并表示感谢。

编　者

1992.3　于南京大学

目　录

第一章　程序设计方法与常用算法

1.1　程序设计与算法

1.1.1　算法

算法就是解决问题方法的精确描述。但是，并不是所有问题都有算法，有些问题经研究可行，则相应有算法；而有些问题不能说明可行，则表示没有相应算法。但这并不是说问题没有结果。例如，猜想问题，有结果，然而目前还没有算法。上述所谓“可行”，就是算法的研究。

1．算法的性质

（1）解题算法是一有穷动作序列。

（2）动作序列仅有一个初始动作。

（3）序列中每个动作的后继动作是确定的。

（4）序列的终止表示问题得到解答或问题没有解答。

2．待解问题的表述

待解问题表述应精确、简练、清楚。使用形式化模型刻划问题是最恰当的。例如，使用数学模型刻划问题是最简明、严格的，一旦问题形式化了，就可依据相应严格的模型对问题求解。

3．算法分类

根据待解问题刻划的形式模型和求解要求，算法可以分成两大类：数值的和非数值的。数值的算法是以数学方式表示的问题求数值解的方法。例如，代数方程计算、矩阵计算、线性方程组求解、函数方程求解、数值积分、微分方程求解等。非数值的算法通常为求非数值解的方法。例如，排序查找、模式匹配、排列模拟、表格处理、文字处理等。将算法分成两大类将有利于算法的设计。

4．算法设计

算法设计的任务是对各类具体问题设计良好的算法及研究设计算法的规律和方法。常用的算法设计方法：数值算法如迭代法、递归法、插值法等；非数值算法如分治法、贪婪法、回溯法等。

5．算法分析

算法分析的任务是对设计出的每一个具体的算法，利用数学工具，讨论各种复杂度。以探讨某种具体算法适用于哪类问题，或某类问题宜采用哪种算法。算法的复杂度分时间复杂度和空间复杂度。设问题规模以某种单位由1增至n，研究解决该问题的具体算法。在运行算法时所耗费的时间为$f(n)$（即n的函数），实现算法所占用的空间为$g(n)$（也为n的函数），则称$O(f(n))$和$O(g(n))$为该算法的复杂度。$O(f(n))$与$O(g(n))$描述了算法的有关性能，所以可用来宏观地评价算法的质量。

1.1.2 数据类型的概念

抽象数据类型是研究解决问题的又一个侧面，也就是算法加工处理的对象的形式。

1. 数据

早期认为数据是物质数量的凭据（或表示）。随着人们处理对象的变化，那些抽象概念（如名称、姓名、颜色等）、语言文字、图象信号以及已被认识的自然现象的表示均为数据。

例如，数值计算处理的对象：整数、实数；书报编辑处理的对象：字符、图形；事务处理的对象：报表、文件。所有这些对象都是数据。

2. 数据结构

简单地说，数据结构是数据构造的逻辑表示形式。

从学科意义上说，数据结构是指构造数据的方法和在所构造数据上构造相应操作的方法，以及对这些方法的研究。

3. 数据类型

数据类型本质上定义了一组值组成的集合，以及相应运算的集合。或者说，一个数据类型定义了变量或表达式可以取值的范围，以及可以施于它们的运算。

数据类型的意义：

（1）无论常量、变量、函数或表达式，它们都有且仅有一个类型。

（2）类型决定了变量、函数和表达式的取值集合以及可施于的运算的集合。

（3）每个值属于且仅属于一个类型。

（4）任何常量、变量、函数或表达式所表示的值的类型，都可由其形式或上下文推知。

因此，可以利用类型信息来避免及查明程序中一些无意义的构造并发现错误。而且使程序人员能按照所要解决的问题，通过类型组织数据，而不是根据所使用的计算机。

1.1.3 结构程序设计

1. 程序

程序是对所要解决问题的各个对象和处理规则的描述。或者说是数据结构和算法的描述。因此，有人称：数据结构 + 算法 = 程序。

2. 程序设计

程序设计就是设计、编制和调试程序的过程。

3. 结构程序设计

结构程序设计就是利用逐步精化的方法，按一套程式化的设计准则进行程序的设计。由这种方法产生的程序是结构良好的。所谓“结构良好”是指：

（1）易于保证和验证其正确性。

（2）易于阅读、易于理解和易于维护。

按照这种方法或准则设计出的程序称为结构化的程序。

“逐步精化”是对一个复杂问题，不是一步编成一个可执行的程序，而是分步进行。第一步编出的程序抽象级最高；第二步编出的程序抽象级比第一步低；依此类推，第i步编出的程序抽象级比第$i-1$步低；直到最后，第n步编出的程序即为可执行的程序。

所谓“抽象程序”是指程序所描述的解决问题的处理规则，是由那些“做什么”操作组

成，而不涉及这些操作“怎样做”以及解决问题的对象具有什么结构，不涉及构造的每个局部细节。这一方法的原理就是：对一个问题（或任务），程序人员应立足于全局，考虑如何解决这一问题的总体关系，而不涉及每一局部细节。在确保全局的正确性之后，再分别对每一局部进行考虑。每个局部又将是一个问题或任务。因而这一方法是自顶向下的，同时也是逐步精化的。采用逐步精化方法的优点是：

（1）便于构造程序。由这种方法产生的程序，其结构清晰、易读、易写、易理解、易调试、易维护。

（2）适用于大任务、多人员设计，也便于软件管理。

逐步精化方法有多种具体做法，例如流程图方法，基于过程或函数的方法。

[例 1.1] 求两个自然数，其和是667，最小公倍数与最大公约数之比是120：1（例如，(115，552)，(232，435)）。

[解] 两个自然数分别为m和$667-m$($2\leqslant m\leqslant 333$)。

处理对象：m（自然数），l（两数最小公倍数），g（两数最大公约数）。

处理步骤：对m从2到333检查l与g的商为120，且余数为0时，打印m与$667-m$。

第0层抽象程序：

```
main()
{ int m, l, g ;
  for (m = 2 ; m<=333 ; m++ ) {
    l = lcm(m, 667 - m) ;
    g = gcd(m, 667 - m) ;
    if ((l == g * 120) && (l%g == 0))
      printf("\n%5d%5d", m, 667 - m);
    }
}
```

第二层考虑函数 gcd（最大公约数）、lcm（最小公倍数）的细化。

最大公约数问题是对参数a、b，找到一个数i能整除a与b，i就是gcd的函数值。

```
int gcd(int a, int b)
{ int i ;
  for (i = a ; i>=1 ; i--)
      if(! ((a%i)||(b%i)))
      return(i) ;
}
```

而最小公倍数的计算是：若干个b之和，若能被a整除，则该和便是a、b的最小公倍数。

```
int lcm(int a, int b)
{ int i ;
  i = b ;
  while (i%a) i += b ;
  return (i) ;
}
```

[例 1.2] 正三角形ABC的边BC、CA、AB上分别有点$A1$、$B1$、$C1$，使$AC1=2C1B$，$BA1=2A1C$，$CB1=2B1A$。验证由线段$AA1$、$BB1$、$CC1$所交成的三角形：$A2B2C2$的面积是三角形ABC面积的$\frac{1}{7}$。

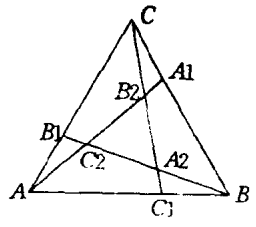

图 1.1 二角形面积比

[解]

处理对象：已知点 A、B、C；辅助点：$A1$、$B1$、$C1$、$A2$、$B2$、$C2$；直线：$AA1$、$BB1$、$CC1$。

处理步骤：输入A、B、C三点坐标值；由点A、B计算$C1$的坐标值；由点 B、C 计算$A1$的坐标值；由点C、A计算$B1$的坐标值；由点A、$A1$计算线段方程的系数；由点B,$B1$ 计算线段方程的系数；由点C、$C1$计算线段方程的系数；由线段 $AA1$、$CC1$ 计算点$B2$的坐标值；由线段 $BB1$、$CC1$ 计算点$A2$的坐标值；由线段$AA1$、$BB1$计算点$C2$的坐标值；验证ABC面积为$A2B2C2$面积的7倍。

第0层抽象程序为：

```
#include<stdio.h>
#include<math.h>
#define eps 1e-6
typedef struct {float x, y;} point;
typedef struct {float a, c;} line;
main()
{point a, b, c, a1, b1, c1, a2, b2, c2;
  line aa1, bb1, cc1;
  readpoint(&a); readpoint(&b); readpoint(&c);
  intermedium(a, b, &c1);
  intermedium(b, c, &a1);
  intermedium(c, a, &b1);
  straightline(a, a1, &aa1);
  straightline(b, b1, &bb1);
  straightline(c, c1, &cc1);
  intersection(aa1, cc1, &b2);
  intersection(bb1, cc1, &a2);
  intersection(aa1, bb1, &c2);
  if (abs(area(a, b, c)-7*area(a2, b2, c2))<eps)
    printf("\nO.K. This proposition is all right.\n");
  else
    printf("\nI find an error on this proposition.\n");
}
```

第一层的分析与细化：

(1) intermedium 计算两点间某一点的坐标。设点为r，原两点为p、q，则 $pr=\frac{2}{3}pq$。于是这一操作的细化为：

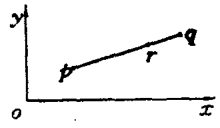

图 1.2 计算线段分点

```
void intermedium(point p, point q, point *r)
{r->x=p.x+2*(q.x-p.x)/3;
 r->y=p.y+2*(q.y-p.y)/3;
}
```

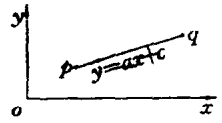

图 1.3 计算直线方程系数

(2) Straightline 为计算线段pq的直线方程 $y=ax+c$ 的系数。这一操作的细化为：

$$\begin{cases} p_y = ap_x + c \\ q_y = aq_x + c \end{cases}$$

（设$p_x \neq q_x$，即pq与y轴不平行）

$$\begin{cases} a = \dfrac{p_y - q_y}{p_x - q_x} \\ c = p_y - a p_x \end{cases}$$

```
void straightline(point p, point q, line * pq)
{ pq->a=(p.y-q.y)/(p.x-q.x);
  pq->c=p.y-pq->a*p.x;
}
```

（3）intersection 为计算两条直线的交点 r。两直线的方程为：

$$\begin{cases} y = a_1 x + c_1 \\ y = a_2 x + c_2 \end{cases}$$

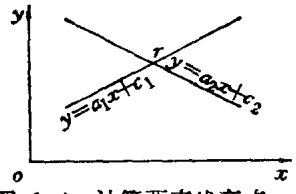

图 1.4 计算两直线交点

其交点为该方程组的解 （设$a_1 \neq a_2$，即两直线不平行）。

$$\begin{cases} x = \dfrac{c_2 - c_1}{a_1 - a_2} \\ y = a_1 x + c_1 \end{cases}$$

这一操作的细化为：

```
void intersection(line pq, line mn, point * r)
{ r->x=(mn.c-pq.c)/(pq.a-mn.a);
  r->y=pq.a*r->x+pq.c;
}
```

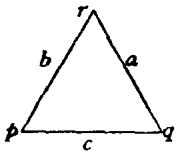

图 1.5 计算三角形面积

（4）area为三角形的面积。设三角形三边长为a、b、c，由海伦公式$A = \sqrt{l(l-a)(l-b)(l-c)}$计算，其中$l=(a+b+c)/2$。这一操作的细化为：

```
float area(point p, point q, point r)
{float l, a, b, c;
  a=distance(q, r);
  b=distance(r, p);
  c=distance(p, q);
  l=(a+b+c)/2;
  return(sqrt(l*(l-a)*(l-b)*(l-c)));
}
```

这里还有一个计算两点距离的函数需要细化。

（5）readpoint 为读一个点坐标值的操作。

```
void readpoint(point * a)
{scanf(" %f %f", &(a->x), &(a->y));
}
```

第二层分析与细化：

经第一层细化后，还剩下（4）中的一个操作：计算两点间距离。由公式

$S=\sqrt{(p_x-q_x)^2+(p_y-q_y)^2}$ 十分容易计算。故 distance 细化为：

```
float distance(point p, point q)
{ return(sqrt((p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y)));
}
```

将它们连在一起，完整的程序便是：

```
#include<stdio.h>
#include<math.h>
#define eps 1e-6
typedef struct {float x,y;} point;
typedef struct {float a, c;} line;

void intermedium(point p, point q, point *r)
{r->x=p.x+2*(q.x-p.x)/3;
 r->y=p.y+2*(q.y-p.y)/3;
}

void straightline(point p, point q, line *pq)
{pq->a=(p.y-q.y)/(p.x-q.x);
 pq->c=p.y-pq->a*p.x;
}

void intersection(line pq, line mn, point *r)
{r->x=(mn.c-pq.c)/(pq.a-mn.a);
 r->y=pq.a*r->x+pq.c;
}

float distance(point p,point q)
{return(sqrt((p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y)));
}

float area(point p, point q, point r)
float l, a, b, c;
  a=distance(q, r);
  b=distance(r, p);
  c=distance(p, q);
  l=(a+b+c)/2;
  return(sqrt(l*(l-a)*(l-b)*(l-c)));
}

void readpoint(point *a)
scanf("%f %f", &(a->x), &(a->y));
}
main()
{point a, b, c, a1, b1, c1, a2, b2, c2;
 line aa1, bb1, cc1;
  readpoint(&a); readpoint(&b); readpoint(&c);
  intermedium(a, b, &c1);
  intermedium(b, c, &a1);
  intermedium(c, a, &b1);
  straightline(a, a1, &aa1);
  straightline(b, b1, &bb1);
  straightline(c, c1, &cc1);
```

```
    intersection(aa1, cc1, &b2);
    intersection(bb1, cc1, &a2);
    incersection(aa1, bb1, &c2);
    if(abs(area(a, b, c)-7*area(a2, b2, c2))<eps)
      printf("\nO.K. This proposition is all right.\n");
    else
      printf("\nI find an error on this proposition.\n");
}
```

以上两例的设计都是“自顶向下”、“逐步精化”的。算法可以如此，数据结构也可以如此。用这种方法设计的程序，使程序易于阅读，利于保证程序的正确，因此具有良好的程序设计风格。

将函数（或过程）调用作为“做什么”操作，而将相应说明看成是“怎样做”操作细节的叙述，这在现代程序设计中是十分重要的方法。

1.2 常用数值计算算法

1.2.1 迭代法

迭代法适用于方程（或方程组）求解，是使用间接方法求方程近似根的一种常用算法。

设方程 $f(x)=0$，该方法将方程表示为等价形式：$x=g(x)$，或一般地将 $f(x)$ 拆成两个函数f_1、f_2，即 $f(x)=f_1(x)-f_2(x)=0$，因而有 $f_1(x)=f_2(x)$。其中$f_1(x)$ 是这样一个函数，对于任意数c，容易求出$f_1(x)=c$的精确度很高的实根。迭代法求解算法如下：

（1）首先选一个x的近似根x_0，从 x_0 出发，代入右面函数，并解方程$f_1(x)=f_2(x_0)$ 得到下一个近似根x_1。

（2）将上次近似根x_1 代入右面函数，并解方程 $f_1(x)=f_2(x_1)$，得到又一个近似根x_2。

（3）重复（2）的计算，得到一系列近似根

$$x_0,\ x_1,\ x_2,\ \cdots x_i,\ x_{i+1},\ \cdots x_n\cdots$$

若方程有根，这数列收敛于方程的根。若满足$|x_n-x_{n-1}|<\varepsilon$，则认为 x_n 是方程的近似根。

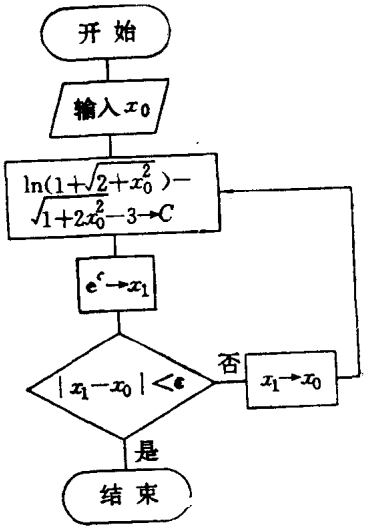

图 1.6 迭代法求方程根

[例 1.3] 用迭代法求方程 $f(x)=\sqrt{1+2x^2}+\ln x-\ln(1+\sqrt{2+x^2})+3=0$的根。

[解] $f_1(x)=\ln x$

$$f_2(x)=\ln(1+\sqrt{2+x^2})-\sqrt{1+2x^2}-3$$

[例1.4] 用迭代法求方程组的根。

$$\begin{cases} ax^2+bx+dy=0 & a,b,d\neq 0 \\ px+qy+ry^2=0 & p,q,r\neq 0 \end{cases}$$

[解]

$$\begin{cases} y=\dfrac{-1}{d}(ax^2+bx) \\ x=\dfrac{-1}{p}(ry^2+qy) \end{cases}$$

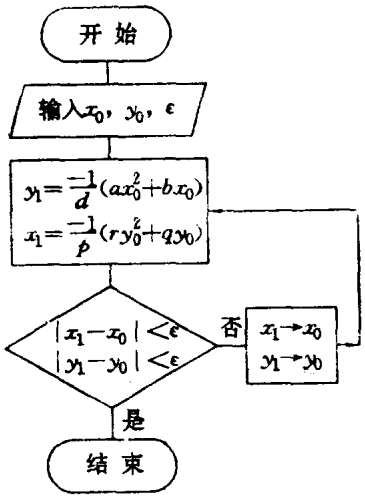

图 1.7 迭代法求方程组的根

使用迭代法应注意：

（1）如果方程无解，数列必不收敛，因而迭代的重复为“死循环”。所以使用迭代算法前应考察方程是否有解。另外，应对重复次数给予一定控制，以防死循环。

（2）当方程有解时，若迭代公式选择不当，或初始近似根选择不当，也会导致迭代失败。例如，迭代公式中出现除数为0、ln0或$\mathrm{tg}\frac{\pi}{2}$等。

（3）有时为了算法的效率，在一定精确度的条件下，利用误差理论，预先确定迭代重复次数，而不采用$|x_n-x_{n-1}|<\varepsilon$的方法。通常采用混合算法。例如，计算$\sqrt{x}$，先利用契比雪夫多项式（二次的）计算$\sqrt{x}$的初始根，再用迭代公式$y_{i+1}=\frac{1}{2}(y_i+\frac{x}{y_i})$迭代两次，便可达到$10^{-12}$的精确度。

1.2.2 递推法

递推法实际上是需要抽象为一种递推关系，然后按递推关系求解。

递推法通常表现为两种方式：一是从简单推到一般；二是将一个复杂问题逐步推到一个已知解的简单问题。这两种方式反映了两种不同的递推方向。前者往往用于计算级数，后者与“回归”配合成为一种特殊的算法——递归法。

由简单推到一般的方法，例如，为得到某项（组）数值，依据前面各项（组）的值推出。通常用于计算级数第n项的值。

[例1.5] 按递增次序生成集合M的最小的n个数。M定义如下：

（1）$1\in M$

（2）$x\in M\Rightarrow y=2*x+1\in M\wedge y=3x+1\in M$

（3）再无别的数属于M

[解] 设n个数在数组m中。

$2x+1$ 与$3x+1$ 均作为一个队列，从两队列中选一排头（数值较小者）送入数组m中，所谓“排头”就是队列中尚未选入m的第一个小的数。这里用p2表示$2x+1$这一列的排头，用p3表示$3x+1$这一列的排头。

[程序]

```
#include<stdio.h>
#define s 100

main()
{int m[s] ;
  int n, p2, p3, i ;
  m[1]=p2=p3=1 ;
  scanf("%d", &n) ;
  for(i=2 ; i<=n ; i++)
```

```
    if(2*m[p2]+1<3*m[p3]+1)
      {m[i]=2*m[p2]+1;
       p2++;}
      else
        {m[i]=3*m[p3]+1;
         p3++;}
  printf("\n");
  for(i=1;i<=n;i++){
    printf("%4d",m[i]);
    if(!(i%10))printf("\n");
    }
}
```

1.2.3 递归法

递归是一种特别有力的工具。不仅在数学中广泛应用，在日常生活中也常常遇到。例如一个画家在作这样一幅画：

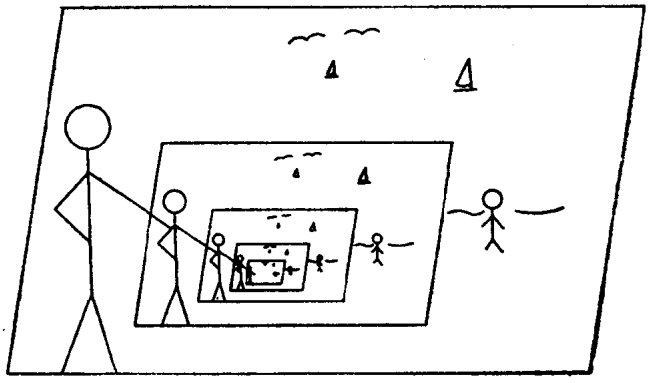

图 1.8 一种递归图形

这便是一种递归的图形。

在数学中几个熟知的递归定义：

(1) $n! = \begin{cases} 1 & \text{当} n=0\text{时} \\ n(n-1)! & \text{当} n\neq 0\text{时} \end{cases}$

(2) 树结构：(a) 0是树（空树）；

(b) 若$t1$和$t2$是树，则  是树。

递归算法包括“递推”和“回归”这两部分。

递推 就是为得到问题的解，将它推到比原问题简单的问题的求解。

例如，$n! = f(n)$，为计算$f(n)$，将它推到$f(n-1)$，即$f(n)=n\cdot f(n-1)$，这就是说，为计算$f(n)$，将问题推到计算$f(n-1)$，而计算$f(n-1)$比计算$f(n)$简单，因为$f(n-1)$比$f(n)$更接近于已知解$0! = 1$。

使用递推应注意：

(1) 递推应有终止之时。例如，$n!$，当$n=0$时，$0! = 1$为递推的终止条件。所谓“终止条件”，就是在此条件下问题的解是明确的，缺少终止条件便会使算法失败。