高等学校规划教材

算法设计与分析

吴哲辉　曹立明　蒋昌俊　编著

煤炭工业出版社

高等学校规划教材

算法设计与分析

吴哲辉　曹立明　蒋昌俊　编著

煤炭工业出版社

(京)新登字042号

内 容 提 要

《算法设计与分析》是计算机类专业的教材。全书共分九章，第一章为算法设计的基本工具和算法分析的内容；第二章为排序的算法设计技术和复杂性分析；第三章为查找；第四章为图算法；第五章以背包问题为线索，介绍了贪心法、回溯法、动态规划、分支限界法等常用算法设计方法；第六章为文本编辑中常遇到的字符串模式匹配的几种算法；第七章为多项式求值、矩阵乘法和向量卷积的一些算法；第八章为NP完全问题；第九章为概率算法、并行算法和符号算法等研究的新方向简介；最后列出了一些课程设计题供选用。

本书是在作者多年来从事教学和研究基础上编写的，是一本比较实用的教材，也是一本较好的参考书。

高等学校规划教材

算 法 设 计 与 分 析

吴哲辉 曹立明 蒋昌俊 编著

责任编辑：胡 玉 雁

*

煤炭工业出版社 出版

（北京安定门外和平里北街21号）

煤炭工业出版社印刷厂 印刷

新华书店北京发行所 发行

*

开本787×1092mm$^1/_{16}$ 印张15$^1/_4$

字数364千字 印数1—2,415

1993年3月第1版 1993年3月第1次印刷

ISBN7-5020-0762-8/TD·705

书号3529 A0211 定价4.00元

前　言

算法研究是计算机科学的核心问题之一。《算法设计与分析》课程的目的是通过对常见而有代表性的问题的算法设计和复杂性分析的介绍，使学生理解和掌握算法设计的主要方法，并培养对算法复杂性的分析能力。因此，本课程的内容对于将要从事计算机类各专业工作的人员来说，都是十分重要的。

国内已出版过若干种算法设计与分析的教材、专著或译著。然而市面上见诸甚少，而且适用的教材也往往订购不到。每当要上这门课之前，教师总要为购买教材而奔忙，并且常常空手而归。可以说，这是促使我们编写这本教材的原动力之一。

其次，通过几年来对算法设计和分析的教学和研究工作，我们也希望在这部教材中写入一点自己的心得和体会，以便向同行专家和读者请教。具体地说，在编写过程中，我们力图使本教材体现下面几方面的特点：

1．《数据结构》是本课程的先导课程和基础，两门课程的内容难免有些交叉。在编写时注意到分清本课程同《数据结构》课的介面，使学生在学习过程中没有过多的内容重复之感。主要做法是：对于数据结构中曾出现过的内容和问题（如排序、查找，图的遍历等），我们把重点放在算法设计技术和复杂性分析方法的阐述上。

2．对于算法设计技术，许多教材都同时采用两种不同的介绍途径。一种是从问题入手，通过对问题的分析提出算法设计的思路；另一种是从方法入手，先提出算法设计的一种思想方法，然后把具体问题的算法设计作为该方法的一个应用实例。在编写这本教材时，全部采用面向问题的写法，即通过对典型问题的分析来介绍算法的设计技术和复杂性分析方法。我们认为，这样编写可能更便于学生掌握和领会。对于将来主要从事计算机应用工作的学生来说，面向问题的写法可能更为实用。

3．在这部教材中，写入了一些有代表性的、能为本科学生所接受的新成果，其中包括算法研究的一些新的发展方向和我们自己的一些研究结果，以便对那些有兴趣进一步学习和研究算法的学生作一些方向性的引导。

以上几点仅是我们的一些想法和尝试。效果如何，还有待进一步的教学实践的检验。

本教材共需要讲授72学时，其中第一、五、八章各需要8学时，第二、四、七章各需要10学时，第三、六、九章各需要6学时。对于不足72学时的教学安排，除了前面指出可把第八章的一部分舍去外，也可把整个第九章作为指导少数学生课外学习的导引。诚然，对于其余各章，任课教师也可根据具体情况进行剪裁和取舍。

本书的第一、五章是由曹立明撰写的，蒋昌俊撰写了第二、三、九章并编写了课程设计题，吴哲辉写了第四、六、七、八章并负责对全书进行校审，是本教材主编。

由于水平所限，书中错误和不当之处一定不少，敬请读者批评指正。

作　者

1992年3月

目　　录

第一章 引　　论

第一节 引　　言

计算机算法是计算机科学和计算机应用的核心，无论是计算机系统、系统软件和解决计算机的各种应用课题都可归结为算法的设计。

算法是一组有穷的规则，它们规定了解决某一特定类型问题的一系列运算。

运算可理解为操作、计算等。用数字计算机一般解决数值计算或非数值计算问题。数值计算算法属于“数值分析”研究对象，如牛顿迭代法，各种线性方程，微分方程解法等等。非数值运算主要讨论排序、搜索、管理、决策、推理、查询和匹配等问题。所以排序和匹配等都可理解为运算。在现实生活中，非数值运算的范围比数值运算范围广泛得多。

算法具有以下五个特点：

1）确定性　算法的每一种运算必须有确切的定义，即每一种运算应该执行何种操作必须相当明确、无二义性。

2）可行性　算法中有待实现的运算都相当基本，它们至少在原理上能由人用纸和笔在有限时间内完成。

3）输入　一个算法有0个或多个输入，它们是在算法开始之前对算法最初给出的量，这些输入取自特定的对象集合。

4）输出　一个算法产生一个或多个输出，它们是同输入有某种特定关系的量。

5）有穷性　一个算法总是在执行了有穷步的运算之后终止。

有穷性是算法的一个十分重要的特征，可以说算法分析就是分析算法的有穷程度——计算机能接受的程度。10亿年也是有穷，这是无限有穷，不能被计算机运算所接受。只满足前四个特点的一组规则称为计算过程。例如操作系统就是一个计算过程。设计操作系统的目的就是控制作业的运行，当没有作业时，这个计算过程并不终止，而是处于等待状态，一直等到一个新的作业进入。所以计算过程可以是个无穷过程。

拟制一个算法一般包括设计、证明和分析三个阶段。设计算法是一种不可能完全自动化的技巧，不同的问题有不同的设计方法，所以必须掌握和熟练运用一些基本设计策略。算法证明要求证明对所有可能的合法输入都能得到正确结果。算法分析则对算法需要多少计算时间和存储空间作定量分析。

第二节 算 法 分 析

要分析一个算法，首先要确定使用哪些运算以及执行此行运算所用的时间，其次是确定能反映出算法在各种情况下工作的数据集，通过使用各种数据配置来执行算法。对一个算法进行全面分析可分为两个阶段，即事前分析和事后测试。事前分析要求出算法的一个时间限界函数；事后测试要求收集算法的执行时间和实际占用空间的统计资料。

定义1-1　如果一个问题的大小是n，解这一问题的某一算法所需的时间为T(n)，它

是n的某一函数。T(n)称为这一算法的“时间复杂性”，当输入量n逐渐增大时，时间复杂性的极限称为算法的“渐近时间复杂性”。

问题大小n是和问题对应的某种测度，如数组的维数，矩阵的阶数，图的边数等。类似于上述定义，也可以定义一个算法的“空间复杂性”和“渐近空间复杂性”。但是在分析一个算法时，我们主要关心算法的时间复杂性，而很少讨论空间复杂性。所以算法复杂性一般指时间复杂性，“时间”通常指通过把问题的输入数据转化为输出结果所需的运算步骤。

计算时间的渐近表示

假设某算法的计算时间是$f(n)$，n是问题的大小，$g(n)$是在事前分析中确定的简单函数，如n^m，$\log_a^n(a>1)$，2^n和$n!$等。

定义1-2 如果存在两个正常数C和n_0，对所有的$n\geqslant n_0$，有

$$|f(n)|\leqslant C|g(n)|$$

则记作

$$f(n)=O(g(n))$$

O为数量级即阶数，$g(n)$是计算时间$f(n)$的一个上界函数，$f(n)$的数量级是$g(n)$。

定理1-1 若$A(n)=a_mn^m+\cdots+a_1n+a_0$是一个m次多项式，则$A(n)=O(n^m)$。

证明 取$n_0=1$，当$n>n_0$时有

$$\begin{aligned}|A(n)|&\leqslant|a_m|n^m+|a_{m-1}|n^{m-1}+\cdots+|a_1|n+|a_0|\\&\leqslant(|a_m|+|a_{m-1}|1/n+\cdots+|a_0|1/n^m)n^m\\&\leqslant(|a_m|+|a_{m-1}|+\cdots+|a_0|)n^m\end{aligned}$$

选取$C=|a_m|+|a_{m-1}|+\cdots+|a_0|$，定理得证。

这个定理表明，变量n的阶数为m的多项式与其最高阶n^m同阶。

定义1-3 如果存在两个正常数C和n_0，对于所有的$n>n_0$，有

$$|f(n)|\geqslant C|g(n)|$$

则记为

$$f(n)=\Omega(g(n))$$

说明$g(n)$是$f(n)$的下界函数。

定义1-4 如果存在正常数C_1，C_2和n_0，对于所有的$n>n_0$，有

$$C_1|g(n)|\leqslant|f(n)|\leqslant C_2|g(n)|$$

则记为

$$f(n)=\theta(g(n))$$

说明$g(n)$既是$f(n)$的上界又是它的下界函数。一个算法的$f(n)=\theta(g(n))$意味着该算法在最好和最坏情况下的计算时间就一个常因子范围内而言是相同的。

比较两个函数的阶数，用下面的定义方式将是很方便的。

假设$\lim\limits_{n\to\infty}f(n)/g(n)=L$，有下列三种情况：

（1）如果$L=a$，a是有限正常数，则$f(n)=\theta(g(n))$；

（2）如果$L=0$，则$f(n)$的阶数比$g(n)$低；

（3）如果$L=\infty$，则$f(n)$的阶数比$g(n)$高。

对于定理1-1，有

$$\lim_{n\to\infty}\frac{A(n)}{n^m}=a_m$$

所以$|A(n)|\leqslant a_m|n^m|$，即$A(n)=O(n^m)$。

如果f(n)=logn，g(n)=n，则

$$\lim_{n\to\infty}\frac{\log n}{n}=\lim_{n\to\infty}\frac{1}{n}=0$$

所以O(logn)<O(n)。

如果$f(n)=n^m$，$g(n)=2^n$，则

$$\lim_{n\to\infty}\frac{n^m}{2^n}=\lim_{n\to\infty}\frac{m\cdot n^{m-1}}{\log 2\times 2^n}=\cdots=\lim_{n\to\infty}\frac{m!}{(\log 2)^m\times 2^n}=0$$

所以$O(n^m)<O(2^n)$。

上面引出的定义和定理可以帮助我们用一些常用的简单函数来分析算法的渐近时间复杂性。

从计算时间上可以把算法分成两类，凡可用多项式来对其时间定界的算法，称为多项式时间算法；而计算时间用指数函数定界的算法称为指数时间算法。以下是几种最常见的对应多项式时间算法和指数时间算法的计算时间函数以及它们之间的比较关系：

（1）$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$；

（2）$O(2^n)<O(n!)<O(n^n)$。

一个计算时间为O(1)的算法，总时间可由一个常数，即零次多项式来限界。上述关系都可用前面阶数比较的极限定义方式来证明。我们已证明了$O(n^m)<O(2^n)$，在证明过程中可以看出，指数时间算法和多项式时间算法在所需时间上非常悬殊。表1-1说明了对不同的问题大小，算法时间复杂性函数之间的差别。

表 1-1 对不同的问题大小常见时间复杂性函数的计算时间

时间复杂性	问题大小n			
	2	8	128	1024
n	2	2^3	2^7	2^{10}
$n\log_2 n$	2	3×2^3	7×2^7	10×2^{10}
n^2	2^2	2^6	2^{14}	2^{20}
n^3	2^3	2^9	2^{21}	2^{30}
2^n	2^2	2^8	2^{128}	2^{1024}
n!	2	5×2^{13}	5×2^{711}	7×2^{8768}

下面一些近似式可给出关于时间的直观概念。

2^{10}步/s≦0.9×2^{16}步/min

≦0.9×2^{22}步/h

≦1.3×2^{26}步/天

≦0.9×2^{35}步/年

≦0.7×2^{42}步/世纪

算法复杂性对计算机科学工作者十分重要，因为存在一个算法并不能保证实际上问题一定能解，这意味着在有效时间内，用计算机计算不可能得到结果。根据上述分析，多项式时间算法是有效的，而指数时间算法只有在n很小时才有效。因此称多项式时间算法为有效算法，而指数时间算法是无效算法。不存在多项式算法求解的问题称为难驾御的问题

(intractable problem)，这类问题只能用近似方法或启发式方法求解。如果能将现有指数时间算法中任何一个算法化简为多项式时间算法，那就取得了一个可喜的成就。

有人认为，随着新一代计算机运算速度的极大提高，关于有效性的讨论将会失去意义。这是一种对算法分析的误解。其实，计算机运行速度的提高，对指数时间算法的影响是十分微弱的。表1-2说明了这个问题。

表 1-2 在规定时间，增加运算速度对于多项式和指数时间算法求解问题产生的影响

时间复杂性	当前速度处理的数据量	速度增长 2^3倍	速度增长 2^7倍	速度增长 2^{10}倍
n	N_1	$8N_1$	$128N_1$	$1024N_1$
n^2	N_2	$2.8N_2$	$11.3N_2$	$32N_2$
2^n	N_3	N_3+3	N_3+7	N_3+10
8^n	N_4	N_4+1	$N_4+2.3$	$N_4+3.3$

表1-2说明了运算速度的增长对多项式时间算法问题的处理起了乘法效应，而对指数时间算法问题的处理只起了有限的加法效应。这进一步说明了算法有效性讨论的重要性。

当前，高速计算机的运算速度可达10亿次/s，约等于2^{30}次/s。我们可以根据表1-1的讨论计算出用这种速度解决一个时间复杂性为2^n，$n=64$的问题需要化多少时间：

$$2^{30}\text{次/s}\fallingdotseq 0.7\times 2^{62}\text{次/世纪}$$

$$2^{64}/0.7\times 2^{62}>2\text{世纪}$$

由此可见指数时间算法的无效性，速度增长对它影响是十分微弱的。

第三节 例 子

例1-1 一个逻辑判断问题：假设有$n(n=3^k)$个外形完全相同的硬币，其中有一个假币，怎样用一个天平把这枚假币找出来？

首先要设计一个算法解决这个问题。最简单的方法是先取定一枚硬币，然后用天平把其它硬币和它逐一比较，则最多称$n-1$次就能找出假币。这种算法的复杂性为$O(n)$。如果把这n个硬币分成三堆，每堆3^{k-1}个，第一次任取两堆放在天平上称，逐次按级减少每堆硬币个数，则最多称$\log_3 n+1$次就能找出假币。可以证明，当$n\neq 3^k$时，用这种算法的复杂性也是$O(\log_3 n)$。因为$O(\log_3 n)<O(n)$，所以第二种算法优于第一种算法，当$n=9$时，我们把9枚硬币编好号码，然后分成(1, 2, 3)，(4, 5, 6)和(7, 8, 9)三堆。先把(1, 2, 3)和(4, 5, 6)两堆硬币放在天平上，算法的执行如图1-1所示。

(1, 2, 3)和(4, 5, 6)比较有三种可能情况：

(1) (1, 2, 3) > (4, 5, 6)；

(2) (1, 2, 3) = (4, 5, 6)；

(3) (1, 2, 3) < (4, 5, 6)。

第一种情况说明(1, 2, 3)中有一个重的假币或(4, 5, 6)中有一个轻的假币。(1, 4)和(2, 5)比较，这也分三种情况：

(1) (1, 4) > (2, 5)；

(2) (1, 4) = (2, 5)；

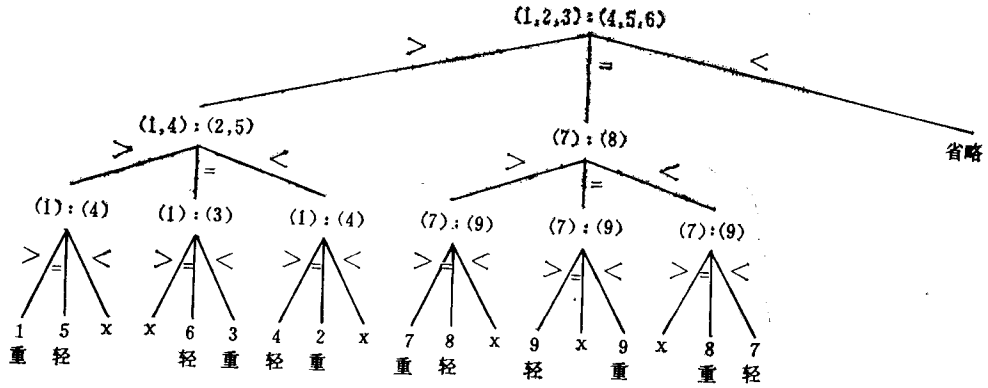

图 1-1 硬币称重算法推理树

（3） (1, 4) < (2, 5) 。

第（1）种情况可以推得1重或5轻；第（2）种情况可推得3重或6轻；第(3)种情况可以推得4轻或2重。再取1和4比较：如果1>4则1是假币，且比真币重；如果1=4则5是假币且比真币轻；1<4不可能。第（2）种情况，可以取1和3比较，1>3不可能；如果1=3则6是假币，且比真币轻；如果1<3则3是假币，且比真币重。第（3）种情况读者可以自己分析。

第（2）种情况，假币一定在7,8,9三枚中,可以取7和8比较：如果7>8则7重或8轻；7=8则9重或9轻；7<8则7轻或8重。这些情况进一步比较和第3种情况，读者可以自己得出结论。

这种算法最多作$\log_3 9+1=3$次比较就可以找出假币，并能确定其轻或重。

例1-2 冒泡排序

对n个元素的序列A(1)，…，A(n)，冒泡排序算法的主要部分是：

```
for i←—1 to n do
    for j←—i+1 to n do
        if A(i)>A(j) then 交换 A(i)和A(j)
    repeat
repeat
```

对每一个i的取值，元素要作$n-(i+1)+1=n-i$次比较或交换运算，i共有n个取值，所以算法的时间复杂性是

$$2\sum_{1\le i\le n}(n-i)=2(1+\cdots+(n-1))=n(n-1)=O(n^2)$$

例1-3 矩阵乘法。

设A和B是两个n阶矩阵，A和B的乘积是C。令$A=[a_{ij}]$，$B=[b_{ij}]$，$C=[c_{ij}]$，则

$$c_{ij}=\sum_{k=1}^{n}a_{ik}b_{kj}$$

对于矩阵C，共有n^2个元素，每个c_{ij}的确定要做n次乘法，$n-1$次加法，即做$2n-1$次运算，所以得到C矩阵共做$n^2(2n-1)=2n^3-n^2$次运算。所以算法的时间复杂性是$O(n^3)$。

例1-4　找出图中两个固定点u和t之间的最大简单路径。

图中一条路径上的点只出现一次的边的序列称为简单路径。任何u和t之间的简单路径组成了图中边集E的子集。下面给出的算法(算法1-1)依次罗列了E的所有子集,其中所有从u到t的路径中最长简单路径都在每次迭代中做了记录。因为E有$2^{|E|}$个子集,所以共有$2^{|E|}$次迭代。该算法（算法1-1）的复杂性是$2^{|E|}$，这是无效算法。

算法1-1　一个最长简单路径算法。

1. MAXP←—0
2. for　所有子集$E' \subseteq E$ do
3. if　E′是从u到t的简单路径

　　then MAXP←—if W(E′)>MAXP then W(E′)

例1-5　旅行售货员问题。

一位售货员在某个区域从一个城市出发周游每个城市一次，然后回到出发地点。他要选择一条路径，保证整个旅行耗费最小。

假设有n个城市，每个城市之间都有道路，那么从某个城市出发经过每个城市一次又回到出发城市的回路共有$\frac{(n-1)!}{2}$个。这种回路称为哈密尔顿回路。旅行售货员问题要求找出一条耗费最小的哈密尔顿回路，其算法复杂性是$\frac{(n-1)!}{2}$,对于n+1个城市的旅行售货员问题，算法复杂是O(n!)。

图1-2是n=4时问题的个例：

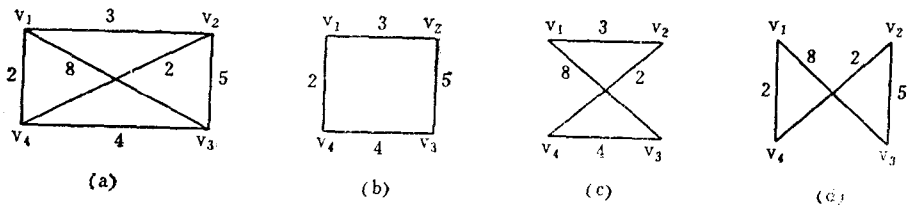

图 1-2　四城市旅行售货员问题

在图1-2 a 中，$(v_1, v_2)=3$,$(v_1,v_3)=8$，$(v_1, v_4)=2$，$(v_2, v_3)=5$,$(v_2, v_4)=2$，$v_3, v_4)=4$。a图中共有$\frac{(4-1)!}{2}=3$个不同的哈密尔顿回路，如图b，c和d所示。b图是售货员要选择的路径。

这些例子初步说明了算法设计和分析的方法。当然有些设计不是很理想的，有待于今后进一步讨论。

第四节　算法描述语言

SPARKS语言是用来写算法的语言，能表示算法本身的基本思想和基本步骤，简明够用，便于阅读且能容易地用人工或机器翻译成其它实际使用的程序设计语言。形式上SPARKS语言与ALGOL和PASCAL语言很接近。

SPARKS语言的基本组成

一、基本数据类型

整型、实型、布尔型和字符型。

如：integer x, y; real a, b; boolean c, d; char e, f分别表示这些数据类型。有特殊含义的标识符作为保留字来考虑，并且用粗体字印出。“；”是语句分隔符。

二、赋值语句

格式：＜变量＞←—＜表达式＞。

左箭头表示把右边的值赋给左边的变量。

有二个布尔值

true 和 false

为产生这两个布尔值设置了逻辑运算符

and, or, not

和关系运算符

$<$, $\leqslant$, $=$, $\neq$, $\geqslant$, $>$

SPARKS使用带有任意整数下界和上界的多维数组。例如，一个n维整型数组可用以下形式说明：integer A(l_1:u_1, …, l_n:u_n)，其下界是l_i，上界是u_i，$1 \leqslant i \leqslant n$，$l_i$和$u_i$都是整数或整型变量。如果某一维的下界$l_i$为1，在数组说明中的那个$l_i$可以省略。例如

integer A(5, 7:20)与integer A(1:5, 7:20)

等效。

三、条件语句

条件语句的格式是：

1. if cond then S_1 else S_2 endif

2. if cond then S_1 endif

其中cond (condition) 是条件，用布尔表达式表示，S_1和S_2是任意组SPARKS语句，endif表示条件语句结束。

3. case语句

```
case
   : cond 1: S1
   : cond 2: S2
      ⋮
      ⋮
      ⋮
   : cond n: Sn
   : else  : Sn+1
endcase
```

其中S_i $1 \leqslant i \leqslant n+1$是SPARKS语句组。case语句是多重条件的简洁表示形式，可以代替使用多重if-then-else语句。

四、循环语句

SPARKS语言提供下列几种循环语句：

1. while语句

```
while cond do
  S
repeat
```

该语句的执行方式如图1-3所示。

2. loop-until-repeat语句

```
loop
  S
until cond repeat
```

该语句的执行方式由图1-4给出，与while语句相比，它保证了至少要执行一次S语句

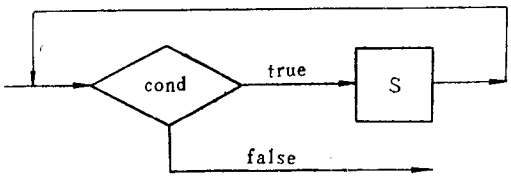

图 1-3 while语句

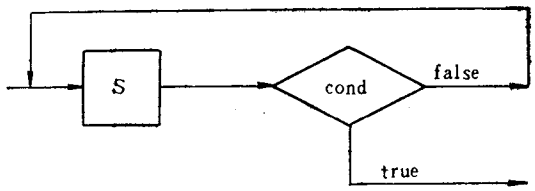

图 1-4 loop-until-repeat语句

3. for循环语句

```
for vble←start to finish by increment do
        S
repeat
```

当increment = 1时 by increment可省去。它的等价语句是：

```
while (vble-finish) * increment≤0 do
  S
  vble← vble + increment
repeat
```

4. loop-until-repeat语句的简化形式

```
loop
  S
repeat
```

它的含义如图1-5所示。

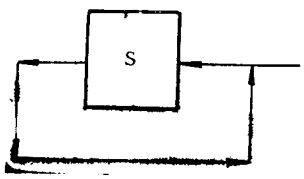

图 1-5 loop-repeat语句

从形式上看，此语句会导致无限循环，因此必须设置一个出口退出循环。一般可在S中使用如下三种语句退出循环：

1) go to语句

```
go to label
label: S1
```

2) exit 它的作用是将控制转移到最内层循环语句的第一条语句。exit能有条件或无条件地使用。如

```
loop
```

S_1
if cond then exit endif
S_2
repeat

其执行情况见图1-6。

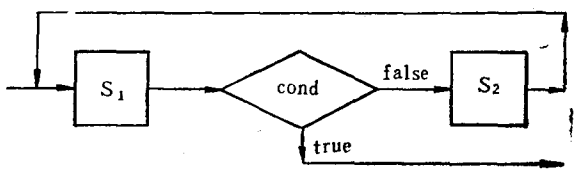

图 1-6　带有exit的loop-repeat语句

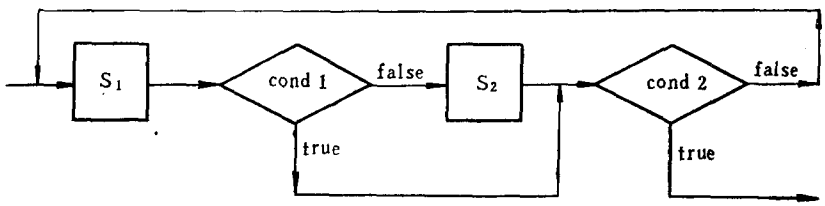

图 1-7　cycle语句

3) cycle　它的作用是将控制转移到包含它的最内层循环语句的结束短语处。如

loop
S_1
if cond1 then cycle endif
S_2
until$_1$ cond$_2$ repeat

五、输入、输出和注解

SPARKS语言的输入和输出只采用两个过程：

read（<参数表>）；print（<参数表>）

首尾均带有双斜线的注解可以放在程序中的任何地方，说明某段程序的主要意思。如

//注解内容//

六、过程

SPARKS过程有如下形式：

procednre　过程名（<参数表>）
　　<说明部分>
　　S
end　过程名

一个SPARKS过程可以是一个纯过程，也可以是一个函数。在函数中，返回值由放在紧接return的一对括号中的值表示。例如：

return（<表达式>）

end的执行意味着执行一条没有值与其相联系的return语句。为了停止程序的执行，可使用stop语句。

一个过程包含三种类型的变量：局部变量，全程变量和形式参数。形式参数在运行时由调用语句中对应位置的实在参数所代替。

在过程中常常涉及对其它过程的调用。对函数过程的调用采用return，对纯过程的调用则采用call语句。如果一个过程包含对自身的调用，则称为直接递归，如果一个过程调用另一个过程，而这另一个过程又调用原来的过程，则称其为间接递归。

例1-6 写一个求数组A(1:n)中最大元素的过程。

算法1-2 求数组最大值

```
procedure MAX(A,n,j)
      // 置j,使A(j)是A(1:n)中的最大元素,n>0. //
      max←—A(1); j←—1
      for  i←— 2 to n do
          if A(i)> max then max←—A(i); j←—i;
          endif
      repeat
end MAX
```

此过程的完整说明是：

```
procedure(A,n,j)
      global real max
      parameters integer j,n;real A(1:n)
      local integer  i;
```

因为SPARKS是为了表示算法，所以过程说明不必如此详尽。只须

```
procedure MAX(A,n,j)
    global max
    integer j,i,n
```

就一个程序语言而论，SPARKS语言是不完整的，但用它描述算法已足够了。因为算法只需叙述解决问题的思路，而不是计算机可直接执行的程序，描述算法有时还使用自然语言或常见的数字表示。带有自然语言或常见数学表示的SPARKS称为拟SPARKS。常用的算法描述语言还有类Pascal语言等。

第五节 设计与分析算法的基本工具

一、递归、迭代与分治

很多问题的数学模型或算法设计方法是递归的。用递归过程描述它们自然明了，简单易读，并且容易证明算法的正确性。但是，在运行时间上由于过程调用和隐式栈管理方面的消费，很自然地会考虑用迭代法代替它。

下面用几个例子说明递归和迭代方法。

例1-7 斐波那契（Fibonacci）级数

意大利数学家斐波那契提出了一个问题：在一年开始时围场里有一对兔子，雌兔每月

生一对新兔子。第二个月开始，每对新兔子也是每月产一对兔子。在一年后围场中有多少对兔子？

假设 F(n) 表示第 n 个月后围场中兔子的对数，$F^{大}_{(n)}$ 表示 n 月后大兔子对的数目，$F^{小}_{(n)}$ 表示 n 月后小兔子对的数目。F(n) 称为斐波那契数。表 1-3 是半年的兔子对数变化情况。

表 1-3 半年的斐波那契数

n	0	1	2	3	4	5	6
$F^{大}_{(n)}$	0	1	1	2	3	5	8
$F^{小}_{(n)}$	1	0	1	1	2	3	5
F(n)	1	1	2	3	5	8	13

因为第 n 个月大兔子对数是上个月大兔子对数加上小兔子长大成的对数；每个月小兔子对的数目等于上个月大兔子对出生的数目。所以

$$F^{大}_{(n)} = F^{大}_{(n-1)} + F^{小}_{(n-1)}$$

$$F^{小}_{(n)} = F^{大}_{(n-1)}$$

因为

$$F(n) = F^{大}_{(n)} + F^{小}_{(n)}$$

所以

$$F^{大}_{(n)} = F(n-1)$$

$$F^{小}_{(n)} = F(n-2)$$

所以

$$F(n) = F(n-1) + F(n-2) \qquad (n \geqslant 2) \qquad (1\text{-}1)$$

显然F(0)=1，F(1)=1，我们可得到斐波那契级数1，1，2，3，5，8，13，…，F(13)=377，即一年后围场中有377对兔子。

数字表达式（1-1）可自然地得出递归的SPARKS过程。

算法1-3　斐波那契级数

```
procedure F(n)
        // 返回第 n 个斐波那契数 //
        integer n
        if n≤1 then return(1)
                else return (F(n-1)+F(n-2))
        endif
end F
```

算法 1-3 在计算时间上效率是很差的，在过程中，很多值被重复计算多次。例如要求 F(5)，则F(3)被重复计算 2 次，F(2)重复计算 3 次，F(1) 重复计算 5 次，因此在写计算机算法时必须对它作进一步改进。

算法1-4　算法1-3的改进算法