课程主要内容

（一）典型问题

1. 排序
2. 查找（检索）
3. 周游
4. 优化问题

（二）算法设计方法

1. 分治法
2. 贪心法
3. 动态规划
4. 图的检索与周游方法
5. 回溯法与分枝限界法

（三）算法复杂度分析

1. 时间复杂度
2. 空间复杂度

（四）一类待解决的问题：NP 完全问题

（五）近似算法、随机算法简介

参考文献

1. **余祥宣，崔国华，邹海明，“计算机算法基础”，华中科技大学出版社，1998。**

2. 朱洪，陈增武，段振华，周克成，“算法设计与分析”，上海科学技术文献出版社，1989。

3. 顾立尧，霍义兴，“算法设计与分析的理论和方法”，上海交通大学出版社，1989。

4. 张益新，沈雁，“算法引论”，国防科技大学出版社，1995。

5. 吴哲辉，曹立明，蒋昌俊，“算法设计与分析”，煤炭工业出版社，1993。

6. 卢开澄，“计算机算法导引－－－设计与分析”，煤炭工业出版社，1993。

7. 郑宗汉，郑晓明，“算法设计与分析”，清华大学出版社，2005。

8. 吕帼英，任瑞征，钱宇华，“算法设计与分析”，清华大学出版社，2006。

9. Robert Sedgewick, Philippe Flajolet (冯舜玺，李学武，裴伟东译),“算法分析引论（An introduction to analysis of algorithms）”，机械工业出版社，2006。

10. 周培德，“算法设计与分析”，机械工业出版社，2006。

11. 霍红卫，“算法设计与分析”，西安电子科技大学出版社，2005。

12. 王晓东，“计算机算法设计与分析”，电子工业出版社，2005。

13. **E. Horowitz, S. Sahni, “Fundamentals of Computer Algorithms”, New York: Computer Science Press, Pitman, Inc., 1978.**

14. **Sara Baase, Allen Van Gelder, “Computer Algorithms: Introduction to Design and Analysis (Third Edition)”, Higher Education Press & Pearson Education Asia Limited., 2001.**

15. **Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest,**

Clifford Stein, “Introduction to Algorithms (Second Edition)”, Higher Education Press & The MIT Press, 2002. **（有中译本）**

开放课程：http://v.163.com/special/opencourse/algorithms.html

16. D. E. Knuth, “The Art of Computer Programming (Third Edition): Fundamental Algorithms (Vol. 1)”, Addison-Wesley, 1998; 清华大学出版社（影印），2002.

17. D. E. Knuth, “The Art of Computer Programming (Third Edition): Sorting and Searching (Vol. 3)”, Addison-Wesley, 1998; 清华大学出版社（影印），2002.

18. A. Levitin, “Introduction to The design & Analysis of Algorithms”, Pearson Education Asia Limited and Tsinghua Unversity Press, 2003.（有中译本）

19. Gilles Brassard and Paul Bratley, “Fundamentals of algorithmics”, Pearson Education Asia Limited and Tsinghua Unversity Press, 2005.

20. **Jon Kleinberg and Eva Tardos, “Algorithm Design”, Pearson Education Asia Limited and Tsinghua Unversity Press, 2006**.（有中译本）

21. R. C. T. Lee, S. S. Tseng, R. C. Chang, Y. T. Tsai, “Introduction to the Design and Analysis of Algorithms”, The McGraw-Hill Education and China Machine Press(机械工业出版社), 2007。（有中译本）

22．Franco P. Preparata, Michael Ian Shamos, “Computational Geometry: An Introduction”, Springer-verlag, 1985.

23. **M H Alsuwaiyel, Algorithms: Design Techniques and Analysis (Revised Edition), World Scientific Publishing Co. Pte. Ltd., 2016**.

第一章　引论

1. 算法

算法是众所周知的一个术语。设计一种算法的目的，总是为了解决一类特定的问题。譬如，欧几里德算法给出两个整数的最大公因子。

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，还是计算机的各种应用课题(例如，模式识别，图像处理，计算机图形学，计算层析成像等)都可归结为算法的设计。

算法设计策略（strategy）是算法学习的重点。应用策略写成具体规范的高效算法是我们的最终目标。许多人常常感慨地说，学过的东西总是在给别人讲述的时候才能真正弄明白。同理，解决问题的策略只有在写成计算机算法，也就是说，在给计算机“讲明白”之后，你才会有透彻的理解，也才能真正应用起来。所以, 具体编写算法不仅能解决特定的问题，还有助于形成缜密细致地思考问题的习惯。

另外，评估一个算法的好坏，即对算法进行分析，也是一个复杂的过程，也需要策略，也有精致的元素值得欣赏。

总之，学习计算机算法，不仅从应用的观点来看是必要的知识储备, 而且对于提高个人的科学素养来说，也是极有益的心智训练。

1.1 算法的概念

算法是一个相当基本的概念，要给它一个准确的形式化定义是十分麻烦的（参考书[2]，形式地说，任何一个对所有有效输入总要停机的图灵机是一个算法。）。从本课程涉及的内容来讲，这样的定义是不必要的。我们宁愿选择下列非形式的一种描述：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（注：这儿的运算指计算机所能提供的各种基本操作）。简言之，算法是问题的过程化解。

例 1.1 （求最大公约数的欧几里得算法）给定两个正整数m和n，$n<m$，求它们的最大公约数，即求能整除m和n的最大正整数。

解：欧几里得算法可描述如下：

E_1：（求余数）以n去除m得余数r，（我们知道$0\le r<n$）。

E_2：（验证$r=0$?）若$r=0$，算法结束，n即为所求；否则（变量递减）置$m\leftarrow n$，$n\leftarrow r$，转E_1。

算法应具备以下五个特性：

（1）输入：一个算法有 0 个或多个输入，它们是对算法给出的初始量。一组完整的输入取自特定的集合，该集合由算法将要解决的问题所确定，称为问题定义域。

例如，（排序）给n个整数排序。问题定义域为

$$\{(a_1,\ a_2,\ \cdots,\ a_n):\ a_i\in\mathbb{Z},\ i=1,\ \cdots,\ n\}$$

其中的元素$(a_1,\ a_2,\ \cdots,\ a_n)$是向量（输入形式为数组）

（2）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

定关系的量或其他形式的结果。

例如 整数排序。输出为

$$(a_{i_1}, a_{i_2}, \cdots, a_{i_n})$$

其中$i_1, i_2, \cdots, i_n$是$1, 2, \cdots, n$的一个置换使得

$$a_{i_1} \le a_{i_2} \le \cdots \le a_{i_n}$$。

（3）确定性：我们一般考虑确定性算法，要求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，无歧义性，例如，不允许“将 6 或 7 与 x 相加”，“取整数n的一个因子”，“从集合S中任取一个元素”之类的运算出现在算法中。

（4）可行性：算法中有待实现的运算都相当基本，即每种运算在原理上必须能被计算机在有限的时间内完成。例如，“两个无理数的精确和”这种运算就不是可行的。

（5）有穷性：一个算法总是在执行了有限步的运算之后终止。

凡是一个算法，都必须满足以上五条特性。只满足前四条特性的一组规则称为计算过程。操作系统就是一个典型的计算过程，而不是算法。操作系统的功能是控制作业的运行，当没有作业时，系统并不终止，而是处于等待状态，等待新的作业进入。所以操作系统从原理上讲永不中止，不具备有穷性。

可以验证，欧几里得算法满足上述五个特性。

既然一个算法是用来求解一种特定类型的问题，为进一步论述起见，我们需要给出问题的形式化描述。

一个问题P是由无穷多个实例组成的一个类。其中每个实例由问

题定义域上的某个实体（即输入）以及对实体的提问（或加工要求）组成。实体和输入，是分别从问题和算法的角度来指称同一个东西。

例 1.2 整数乘法问题：问题定义域是全体整数偶对，实例的实体是数对(a, b)，提问：它们的积是多少？该问题的一个实例是：$2\times 4=$？

请注意，个别的实例不认为是一个问题。

一个算法A称为解算问题P，或称A是问题P的一个算法，是指如果把P的任一实例的实体作为A的输入，A在有限步之内总输出一个关于此实例的正确答案。一个问题P称为算法可解的，如果存在一个算法解算P。

一个问题是算法可解的，是否实际可解呢？回答是并不尽然。因为解算这个问题的算法在现有的计算机上可能需要极长的时间（比方说 10 亿年），以至于无法接受，也即它们不是有效算法。这就导致对一个算法的定性与定量分析的分界线，定性是指一个算法的有穷性，即它是否总在有限步内终止，至于究竟多少步则不论，反正只要是有限步就可以了。定量是一个算法的有效性，即它到底在多少步内终止。这个步数必须要有限度，必须是可以接受的，现实可行的。对于后者的研究，就是所谓计算复杂性的问题。

算法的设计和分析主要考虑两个方面：

（一）给定一个问题，如何设计出解算它的有效算法。

（二）分析和判断一个算法的质量，主要指效率的度量。

关于第一方面的问题，随后各章将陆续介绍一些常用的设计策略。下面来讨论一下分析算法的准则和技术。

1.2 分析算法的几条准则

评价一个算法的性能质量，通常可参照下列五条准则：

（1）正确性：称解算某问题的一个算法是正确的，如果对任给的一个有效输入（即问题定义域上的一个实体），该算法总在有限时间内给出相应实例的正确答案。一个算法的正确性包括对问题的解法在逻辑上是正确的，且指令序列具有确定性，可行性，有穷性，我们主要关心前者。数学归纳法常用来证明算法的正确性。为了证明算法的正确性，必须对输入的有效性和输出的正确性作严格的定义。为证明一个大型程序的正确性，可以试图将这个程序分解成一些较小的程序段，通过证明所有这些较小的程序段是正确的来证明整个程序是正确的。

（2）简单性：要求算法易于理解，易于编程，易于调试。

（3）复杂度：包括时间复杂度和空间复杂度，即在计算机上运行算法时所需时间和存储空间的量度。显然，复杂度较低的算法效率较高。

（4）最优性：算法A是问题P的最优时间算法是指：解算问题P的所有算法中，A执行的基本运算次数最少。证明算法的时间最优性的一个常用方法是，先从理论上证明一个问题所需基本运算次数的下界，如果一个算法执行的基本运算次数恰是这个下界的值，则该算法是时间最优的。空间最优性也可同样分析。需要指出，空间最优性与时间最优性往往难以并存于一个算法中，根据实际情况，可以在两种资源的使用方面实施均衡。

（5）可修改可扩展性：如果问题A是解算问题P的一个算法，为了解算一个与问题P相似的问题$\overline{P}$，希望对A稍作改动就可正确运行，若算法A满足这一点，则说算法A的可修改性好；否则，如果要解决问题$\overline{P}$，不得不对A作多处重大修改，那还不如重新写一个算法来得简便，则说A的可修改性不好。同样地，有时希望扩大算法A所解决的问题范围，即要给A增加一些功能，如果只要对A稍作修改即可，则称A的可扩展性好。反之，A的可扩展性不好。

以上说法有些术语（例如，什么是基本运算）的确切含义并未解释，待后将在有关论述中作进一步的澄清。

就理论上严格的分析来讲，我们主要关注算法的正确性，复杂度和最优性这三条评价准则。当然正确性是必要的，复杂度是评价算法的最基本量度。

下面着重介绍分析算法复杂度的基本原理，经常仅限于讨论时间复杂度。空间复杂度的分析作类似处理。

算法的时间复杂度，即算法的运行所需的时间，就输入而言，与两个因素有关：

（1）输入的规模大小，例如，在排序问题中，给n个整数排序，n即表征输入的规模。一般来说，输入规模越大，运行时间越长。

（2）即使输入规模相同，算法运行的时间还与输入的具体情况有关，例如，给100个几乎有序的整数排序应比给100个杂乱无章的整数排序所需时间少。

综合上述两个因素，算法的时间复杂度以函数形式来表示，自变

量为表示输入规模的参数，函数值取相同规模的输入下算法运行所需的最大时间或平均时间。这两个函数分别刻画算法的最坏时间复杂度和平均时间复杂度。

输入规模通常用一个或多个非负整数来表示。例如，排序问题中用要排序的整数个数n表示输入规模，可用图模型表示的问题用图的结点数n和边数m表示输入规模。为简化起见，下面仅以输入规模是一个整数为例来作详细阐述。

表 1.1 问题和输入规模举例

问题	规模
1. 在一个表中搜索元素 x	表中元素的个数
2. 两个方阵相乘	方阵的维数
3. 整序一个表	表中元素的个数
4. 遍历一个二叉树	树中顶点（结点）个数
5. 解一线性方程组	方程个数或未知数个数或两者兼有

以X表示问题的输入，X本身也是一个集合，例如排序问题的输入由若干个可比较大小的元素组成，记$|X|$为输入规模。算法A的**最坏时间复杂度**用$WT_A(n)$表示，定义为

$WT_A(n) := \max\{t$:存在输入X，$|X|=n$，且算法A对输入X的运行时间是$t\}$.

算法A的**平均时间复杂度**用$MT_A(n)$表示，定义为

$$MT_A(n) := \sum_{I\in \mathrm{P}_n} p(I)t(I),$$

其中P_n表示输入规模为n的实例集D_n的一个剖分，使得P_n中任何元素I作为D_n的子集满足：I中所有实例的输入在算法A下运行的时间相同。以$t(I)$表示该时间。$p(I)$是I中元素在D_n中出现的概率。

将规模为n的输入集X_n看作随机变量集，算法运行时间$t(X_n)$则可看作一个随机变量，算法的平均时间复杂度正是$t(X_n)$的期望，即

$MT_A(n)=E(t(X_n))$。

在不致引起混淆的情况下，$WT_A(n)$或$MT_A(n)$常简记为$T_A(n)$或$T(n)$。

上面讨论了算法的时间复杂度与输入的关系。下面来讨论算法运行时间的度量标准，即如何确定算法所包含的指令的运行时间。

算法运行时间的度量法则，应该与实际使用的计算机，程序设计语言，程序员的编程技术无关，还应该与许多实现细节或称为“薄记”操作（例如，循环下标计数，数组下标计数，或设置指针等）无关，但是它又必须能反映算法实际执行时间方面的信息。

为此，我们选定一些特定的操作，称为**基本操作**，它对被研究的问题（或对被讨论的算法类）来说是基本的。基本的含义包括这些操作是解算这一问题的关键操作，将会在算法中反复使用，操作次数将会随着输入规模增大而增大。而“薄记”操作则不一样，它也许保持常数或以较小的速度增加。因此，弃置“薄记”工作量于不顾，只计算算法所执行的基本操作次数。若这算法所执行的操作总数粗略地与基本操作数成比例（当输入规模比较大时），则这一指定是合理的。

更精确地说，如果基本操作的指定是合理的，那么假设算法对输入X做$T(X)$次基本操作，则操作总数介于 $C_1T(X)$ 与 $C_2T(X)$ 之间，而实际执行时间介于$C_3T(X)$ 与 $C_4T(X)$ 之间, 这里C_1，C_2，C_3，C_4是常数，且$C_1 < C_2$，$C_3 < C_4$。C_1，C_2仅依赖于这个算法的薄记方法，C_3，C_4还依赖于实现该算法的计算机，程序设计语言以及程序员，而都与 x 的规模无关。因此，只要基本操作选取得合理，算法所执行的基本操作数就是对算法运行时间的一个好的度量。对于解算同一问题的若干算法来说，只要选取的基本操作相同，也就有了客观地比较它们优劣的准绳。

表 1.2 几个问题的基本操作

问题	基本操作
1. 在一个表中搜索元素 x	比较（x 与表中一元比较）
2. 两个实矩阵相乘	实数乘法或实数的乘法和加法
3. 整序一个表	比较（表中两元比较）
4. 遍历一个二叉树（链接结构）	访问一个链接，置一个指针

采用上述分析算法时间复杂度的原理，来考察两个例子。

例 1.3 检索问题

Procedure SEQUENSESEARCH(A, n, j, x)

//$A(1:n)$是一n元数组，x是任一给定的元素，判断x是否出现//

//在数组A中。若是，置j，使$A(j)=x$；否则，置$j=0$。//

$j \leftarrow n$

while $j \geq 1$ and not $A(j) = x$ do

$j \leftarrow j - 1$

repeat

end SEQUENSESEARCH

分析：该检索问题的算法中，基本操作是两元比较，计算算法运行时间，只需计数比较次数。

算法的平均时间复杂度: 输入A, x能够依它在A中何处出现而分类，即有$n+1$个输入类要考虑，令I_i $(1 \leq i \leq n)$ 表示x在A中第i个位置的输入类，I_0表示x不在A中的输入类。显然，算法对输入类I_i $(1 \leq i \leq n)$ 中输入的比较次数是:$t(I_i) = n+1-i$，对输入类I_0中输入的比较次数是:$t(I_0) = n$。为计算平均比较次数，必须给出x在A中的概率和x在A中每个位置的概率。设x在A中的概率为q, 且假设x在A中每个位置出现的概率相同，则

$$p(I_i) = q/n \ (1 \leq i \leq n); \ p(I_0) = 1 - q.$$

所以平均时间复杂度为

$$MT(n) = \sum_{i=0}^{n} p(I_i)t(I_i) = q(n+1)/2 + (1-q)n.$$

另一方面，容易看出，I_0是该算法最坏情况下的输入类，所以，

$$WT(n) = t(I_0) = n.$$

讨论：(1) 若$q = 1$, 即x必在A中，则$MT(n) = (n+1)/2$，在平均意义下需要检索数组的一半。(2) 若$q = 1/2$, 即x在A中与x不在A中的概率相等，则$MT(n) = (3n+1)/4$, 在平均意义上需要

检索数组的四分之三；(3) 若 $q=0$，即 x 必不在 A 中，则 $MT(n)=n$, 即整个数组都要检索到。此时，平均时间复杂度达到了算法的最坏时间复杂度。

例 1.4 矩阵乘法问题

```
procedure MULMATRIX(A,B,C,n)
// 求矩阵 A(1:n, 1:n) 与矩阵 B(1:n, 1:n) 的乘积 C(1:n, 1:n)).//
integer i, j, k
for i ← 1 to n do
 for j ← 1 to n do
    C(i, j) ← 0
  for k ← 1 to n do
    C(i, j) ← C(i, j) + A(i, k)B(k, j)
  repeat
 repeat
repeat
end MULMATRIX
```

分析：该算法中，基本操作是两实数的乘法。对 C 的每一元素 $C(i, j)$ 需做 n 次乘法，C 有 n^2 个元素，所以共需 n^3 次乘法。由此看来，该算法的运行时间只与输入规模有关，而与输入的具体情形无关，这种现象被称为是**数据无关的**。此时，算法的平均时间复杂度与最坏时间复杂度相同：$WT(n)=MT(n)$，记为 $t(n)$。本例中，

$t(n)=n^3$。

1.3 时间复杂度的渐近表示与算法分类

前述两例中，算法的时间复杂度都可表示为简单的多项式函数。但很多情况下，我们很难写出算法时间复杂度函数的显式表达。为此，我们采用一些简单的整数函数来渐近表示算法的时间复杂度，这样的整数函数诸如:

$$\log n,\ n,\ n\log n,\ n^k(k\geq 2),2^n,\ n!,\ n^n$$

等等都是量度算法复杂度的标准函数。下面来说明渐近表示的含义。

定义 1.1 设$T(n)$是某算法的时间复杂度，$g(n)$是一标准函数，若存在正常数c_0和正整数n_0，使得对所有的$n\geq n_0$，都有

$$T(n)\leq c_0 g(n) \tag{1.3}$$

则记

$$T(n)=O(g(n)) \tag{1.4}$$

称$g(n)$是$T(n)$的渐长率的一个上界；若存在正常数c_1和正整数n_1，使得对所有的$n\geq n_1$,都有

$$T(n)\geq c_1 g(n) \tag{1.5}$$

则记

$$T(n)=\Omega(g(n)) \tag{1.6}$$

称$g(n)$是$T(n)$的渐长率的一个下界；若既有$T(n)=O(g(n))$，又有$T(n)=\Omega(g(n))$，则记

$$T(n)=\Theta(g(n)) \tag{1.7}$$

称$T(n)$与$g(n)$的渐长率是同阶的。注意，这里的渐长率与通常所

说的变化率不是一回事。

例 1.5 设$T(n)=3n^4\log n+n^3\log^3 n$，则$n^4$是它的一个下界，$n^5$是它的一个上界。$n^4\log n$与它同阶。

渐近记号的性质：

(1) $O(O(g(n)))=O(g(n))$；

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

$\Theta(\Theta(g(n)))=\Theta(g(n))$.

(2) $O(g(n))+O(g(n))=O(g(n))$;

$\Theta(g(n))+O(g(n))=\Theta(g(n))$;

$\Omega(g(n))+O(g(n))=\Omega(g(n))$.

在实际分析算法时，我们当然希望能找到$g(n)$使得$T(n)=\Theta(g(n))$，不得已求其次，找尽量小数量级的$g(n)$,使$T(n)=O(g(n))$, 以$g(n)$作为$T(n)$的一个上界估计（或限界）。在实际应用中，我们总是希望算法复杂度不要超过某个数量级就可满意，所以给出一个恰当的上界估计也就够了。

从时间复杂度上，可以把通常遇到的算法分成两类：**多项式时间算法**和**指数时间算法**，分别指以多项式函数限界和指数函数限界的算法。

下面比较一下几种标准函数表示的时间复杂度的数量级：

(1)表示多项式时间复杂度的常见标准函数如下

$$1\ll\log n\ll n\ll n\log n\ll n^2\ll n^k\ \ (k>2)$$

(2)表示指数时间复杂度的常见标准函数如下

$$2^n \ll n! \ll n^n$$

其中“$\ll$”表示远远小于。

说明：指数时间算法只有当输入规模n很小时才可以接受，以时间复杂度为2^n的算法为例，当$n=40$时，$2^{40}=1024^4>10^{12}$，运算次数已相当可观。即使是多项式算法，只有当多项式的次数很低时，才可以接受，因为输入规模往往会达到几千，几万甚至更大。

表 1.3　不同量级复杂度之比较

时间复杂度 $T(n)$	可解决的最大实例规模(1秒) n	可解决的最大实例规模(1天) n	可解决的最大实例规模(1年) n
n	10^9	8.64×10^{13}	3.1536×10^{16}
$n\log n$	39620077	2.11038×10^{12}	6.4114×10^{14}
n^2	31623	9.2952×10^6	1.7758×10^8
n^3	1000	4.4208×10^4	3.1594×10^5
2^n	29	46	54

注：指数时间复杂度增长速度惊人的快，就$T(n)=2^n$来说，一年的运行时间能解决输入规模为 54 的实例。而输入规模从$n=54$增加1到$n=55$时，运行时间将翻一番，增加到两年。当爱因思坦被问到，他所碰到过的最伟大奇迹是什么时，他回答说，是复利的增长。

他的意思就是说，指数增长速度快的不可思议。

通常来说，时间复杂度为$O(n\log n)$的算法才是实际可用的，由此导出下面说法：若问题P存在时间复杂度为$O(n\log n)$的算法，则称它是实际可解的；若解算问题P的所有算法的时间复杂度都是$\Omega(a^n)$，其中，a是某个大于1的常数，则称它是实际不可解的，例如著名的 hanoi 塔游戏问题就是实际不可解的.

例 1.6 Hanoi 塔游戏

有A, B, C三根柱子，n个大小各不相同的盘子从大到小叠放在柱子A上,要求将A上的这n个盘子借助于柱子B以同样的方式叠放在C上。盘子移动的规则如下：(1)每次只准移动一个盘子；(2)在任何时侯，都不允许大盘子叠放在小盘子上。

解：分析：当$n=1$时，直接将这一只盘子从柱子A上移放到柱子C上；当$n>1$时，如果有成功的移动方案的话，由盘子的移动规则，当将柱子A上的最大盘子移动时，B,C柱子必有一个为空，而另一个柱子上从大而小叠放着其余的$n-1$个盘子，由柱子B,C的对称性（指起始B,C都是空的），我们当然选择C为那个空柱子，以使 A上的最大盘子移放在C上。依此事实，将问题转化为三个子问题：

(1)将A上的$n-1$个盘子借助C移放在B上；

(2)将A上的最大盘子移放到C上；

(3)将B上的$n-1$个盘子借助A移放到C上。

这就给出了解决该问题的递归算法。从以上分析可知，该算法是移动盘子次数最少的唯一方法。

算法描述如下：

procedure HANOI(n, A, B, C)

// 将n个盘子从柱子A借助于柱子B移放到柱子C上。//

if $n=1$ then Write(n, A, C) //打印信息，将n号盘子从A//

//上移到C。//

else

call HANOI($n-1$, A, C, B)

Write(n, A, C)

call HANOI($n-1$, B, A, C)

endif

end HANOI

Hanoi 塔问题的基本操作是移动，上述递归算法的时间复杂度$T(n)$满足下列递归式：

$$\begin{cases} T(1)=1, \\ T(n)=2T(n-1)+1,\ n\geq 2 \end{cases} \tag{1.8}$$

可通过一个简单变换来求解该递归式。令

$$t(n)=T(n)+1,$$

则有

$$\begin{cases} t(1)=2, \\ t(n)=2t(n-1),\ n\geq 2 \end{cases} \tag{1.9}$$

故得到

$$t(n)=2^n,$$

从而

$$T(n)=2^n-1.$$

趣闻：约在十九世纪末，在欧洲的珍奇商店里出现了一种称为 hanoi 塔的游戏。这种游戏由于附有推销材料，因而更加流行起来。推销材料中说，布拉玛神庙（Temple of Bramah）里的教士们正在玩这种游戏，他们的游戏结束就标志着世界末日的来临。教士们的游戏装置很简单，一块铜板上面插着左, 中, 右 3 根金刚石针，左针上从大到小叠放着 64 个不同大小的金盘。游戏的目标是把左边针上的 64 个金盘移到右边针上，规则是一次只能移动一个金盘，可以利用中间针作为过渡，但在任何时刻均不允许大盘放在小盘上面。当时人们看了这个介绍材料均很惊恐，以为世界很快要毁灭了。

根据我们上面的论证，这 64 个金盘总共需要移动$2^{64}-1$次，即使每秒能移动1百万次，要完成这个游戏得花费1百万年。

注：常系数线性递归式（1.9）的一般形式是

$$\begin{cases} T(1)=t_1,\ T(2)=t_2,\ \cdots,\ T(k)=t_k \\ T(n)=a_1T(n-1)+a_2T(n-2)+\cdots+a_kT(n-k),\ n\geq k+1 \end{cases} \tag{1.10}$$

其中，$t_1,\ t_2,\ \cdots,\ t_k;\ a_1,\ a_2,\ \cdots,\ a_k$ 是实常数.

式(1.10)可用母函数法求解. 以解式(1.8)为例，构造如下母函数

$$G(x)=\sum_{n=1}^{\infty}T(n)x^n \tag{1.11}$$

上式两边同乘以$2x$得

$$2xG(x)=\sum_{n=1}^{\infty}2T(n)x^{n+1}=\sum_{n=2}^{\infty}2T(n-1)x^{n} \qquad (1.12)$$

式(1.11)减去式(1.12)得

$$(1-2x)G(x)=T(1)x+\sum_{n=2}^{\infty}x^{n}=\sum_{n=1}^{\infty}x^{n}=\frac{x}{1-x}$$

进一步得到$G(x)$的如下展开式：

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

$$=\sum_{n=0}^{\infty}(2x)^{n}-\sum_{n=0}^{\infty}x^{n}=\sum_{n=0}^{\infty}(2^{n}-1)x^{n} \qquad (1.13)$$

由$G(x)$展开式的唯一性推知

$$T(n)=2^{n}-1。$$

一般地，也可以依据下列定理采用待定系数法求解（1.10）。

定理 1.1（参考书[9]）设β是齐次常系数线性递归式

$$T(n)=a_1T(n-1)+a_2T(n-2)+\cdots+a_kT(n-k)， \quad (1.14)$$

的特征方程

$$x^{k}-a_1x^{k-1}-a_2x^{k-2}-\cdots-a_k=0$$

的r重实根，则

$$n^{s}\beta^{n}（0\le s\le r-1）， \qquad (1.15)$$

是满足(1.14)的r个线性无关的解。

定理 1.2 设$qe^{i\theta}$和$qe^{-i\theta}$是齐次常系数线性递归式

$$T(n)=a_1T(n-1)+a_2T(n-2)+\cdots+a_kT(n-k)，$$

的特征方程

$$x^k - a_1 x^{k-1} - a_2 x^{k-2} - \cdots - a_k = 0$$

的r重共轭复根，则

$$n^s q^n \cos(n\theta)，\ n^s q^n \sin(n\theta)（0 \le s \le r-1），\quad (1.16)$$

是满足(1.14)的$2r$个线性无关的解。

定理 1.3 齐次常系数线性递归式

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots + a_k T(n-k)，$$

的解空间是k维的，即（1.14）式的任何解都可以表示成k个形如（1.15）式或(1.16)式所示线性无关解的线性组合，组合系数由初始条件

$$T(1) = t_1,\ T(2) = t_2,\ \cdots,\ T(k) = t_k$$

所确定。

3. 算法语言： SPARKS

为便于表达算法所具有的特性，最好将算法用一种程序语言写成程序的样子。选择语言的原则简单说来，就是够用好用。具体地说，由该语言所写出的每一个合法的句子必须具有唯一的含义，用它写出的算法便于阅读并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，同时要求该语言相当简明，能反映算法本身的基本思想和基本步骤即可，不必太关注细节。为此，我们选用一种符合以上要求的 SPARKS 语言。下面对它作一简单介绍。

（一）数据类型与结构

SPARKS 的基本数据类型包括整型，实型，布尔型和字符型，变量只能存放单一类型的值，它可以用下述形式来说明其类型：

integer i, j; real x, y; boolean a, b; char c, d;

有特殊含义的标识符作为保留字，给变量命名的规则是：以字母起头，不许使用特殊字符，不要太长，不许与任何保留字重复。

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 $A(l_1 : u_1, \cdots, l_n : u_n)$

其中l_i, u_i $(1 \le i \le n)$为整数或整型变量，分别表示第i维的下界和上界。如果某一维的下界l_i为1，在数组说明中，此l_i可以不写出。例如,

integer $A(5, 3:10)$

与

integer $A(1:5, 3:10)$

同义。为保持 SPARKS 语法的简明性，只使用数组作为基本结构单元来构造所有数据对象，而没有引进记录等结构类型。

（二）语句

1. 赋值语句：（变量）←（表达式）

2. 逻辑运算符：and, or, not (布尔值共有两个：true 和 false)

3. 关系运算符：＜，≤，＝，≠，＞，≥

4. 条件语句：

if cond then S_1

else S_2

endif

或

if cond then S

endif

其中 cond 表示条件，$S, S_i (i=1,2)$表示 SPARKS 语句组。

5. case 语句

case

:cond1: S_1

:cond2: S_2

……

:condn: S_n

:else: S_{n+1} (可缺省)

endcase

6. 循环语句

(1) while cond do

S

repeat

(2) loop

S

(until cond) repeat

（3）for vble $\leftarrow$ start to finish (by increment) do

S

repeat

其中 vble 是一个变量，start, finish 和 increment 是算术表达式。一个整型或实型变量或一个常数都是算术表达式的简单形式。子句 by increment 缺省时, increment 自动取$+1$。

补充说明：上述循环体的语句组S中可加入某种检测条件导致一个出口。例如，可在S中加入

if cond then go to label endif

语句，它将控制转移到附标号"label"的语句：label: S。go to label 语句的两种特殊形式是

Exit 和 cycle

例如

loop

S_1

if cond1 then exit endif

S_2

until cond2 repeat

意谓当 cond1=true 时，退出整个循环。而

loop

S_1

if cond1 then cycle endif

S_2

until cond2 repeat

与

loop

S_1

if cond1 then go to label endif

S_2

label: until cond2 repeat

等效

（三）SPARKS 程序

一个完整的 SPARKS 程序是一个或多个过程的集合，第一个过程作为主程序，执行从主程序开始。单个的 SPARKS 过程有以下形式：

procedure NAME([参数表])\

(说明部分)

S

end NAME

一个 SPARKS 过程可以是一个纯过程（又称为子例行程序），也可以是一个函数。在函数中，返回值由放在紧接 return 的一对括号中的值来表示。例如

return([参数表])

其中表达式的值作为函数的值来传送。对于纯过程而言，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，可以使用 stop 语句。

执行任一 SPARKS 过程，譬如过程 A, 当到达 end 或 return 语句时，控制返回到调用过程 A 的那个 SPARKS 过程。如果过程 A 是主程序，控制则返回到操作系统。调用过程 A 的语句为

call A

SPARKS 过程允许递归调用，包括直接和间接递归。

（四）补充说明

输入，输出对拟定算法模型是不必要的，所以 SPARKS 对输入，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，例如

// 检索过程开始//

最后，当用自然语言或数学表示能较好地描述算法模型时，我们也毫不犹豫地这样做。

至此，SPARKS 语言说明完毕。

作业

1.（**石子游戏**）有 $n(n\geq 2)$ 个石子，甲乙两个玩如下规则的取石子游戏：

(1) 甲乙轮流取石子, 甲先取;

(2) 设甲各次取的石子数分别为

$m_{甲}^{1}$，$m_{甲}^{2}$，$m_{甲}^{3}$，…

乙各次取的石子数分别为

$m_{乙}^1,\ m_{乙}^2,\ m_{乙}^3,\ \cdots$

则要求

$$1 \le m_{甲}^1 < n, \qquad 1 \le m_{乙}^1 \le 2m_{甲}^1;$$
$$1 \le m_{甲}^2 \le 2m_{乙}^1,\ 1 \le m_{乙}^2 \le 2m_{甲}^2;$$
$$1 \le m_{甲}^3 \le 2m_{乙}^2,\ 1 \le m_{乙}^3 \le 2m_{甲}^3;$$
$$\cdots$$

直到取完石子为止，最后一次拿到石子的一方赢!

证明：当n是 Fibonacci 数时，乙总有一种策略必赢；而当n是其他整数时，甲总有一种策略必赢。

2.求满足下列递归式的$T(n)$的渐近表示:

(1) $T(n) = T(n-1) + n;$

(2) $\begin{cases} T(1) = T(2) = 1, \\ T(n) = T(n-1) + T(n-2),\ n \ge 3 \end{cases};$

(3) $T(n) = T(\lfloor \sqrt{n} \rfloor) + 1;$

(4) $T(n) = \dfrac{n}{n-1} T(n-1) + 1.$

3. (有序拆分问题) 给定正整数n, 要求写出n的所有有序拆分. n的一个有序拆分是指一个正整数数组$(n_1,\ \cdots,\ n_i)(1 \le i \le n)$, 使得

$$n = n_1 + \cdots + n_i.$$

例如, 3的所有有序拆分为

$$(1,\ 1,\ 1),\ (1,\ 2),\ (2,\ 1),\ (3).$$

请设计求解该问题的算法, 并分析其时间复杂度。