经典计算机科学著作最新修订版

计算机程序设计艺术

第2卷 半数值算法

（第3版）

The Art of Computer Programming

苏运霖 译

[美] DONALD E. KNUTH 著

Addison Wesley

国防工业出版社

National Defence Industry Press

http://www.ndip.com.cn

计算机程序设计艺术

第2卷

半数值算法

（第3版）

[美] Donald E. Knuth 著
苏运霖 译

国防工业出版社

著作权合同登记号　图字:军-2001-019 号

图书在版编目(CIP)数据

计算机程序设计艺术. 第 2 卷, 半数值算法 / (美)克努特(Knuth, D. E.);苏运霖译. —3 版. —北京: 国防工业出版社, 2002.8
ISBN 7-118-02707-3

Ⅰ.计... Ⅱ.①克...②苏... Ⅲ.①电子计算机-算法理论②电子计算机-随机数产生法 Ⅳ.TP301.6

中国版本图书馆 CIP 数据核字(2001)第 078811 号

互联网网页 http://www-cs-faculty.stanford.edu/~knuth/taocp.html 包含本书及相关著作的最新信息。

國防工業出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码　100044)
北京奥隆印刷厂印刷
新华书店经营
*
开本 787×1092　1/16　印张 48½　959 千字
2002 年 8 月第 1 版　2002 年 8 月北京第 1 次印刷
印数:1—4000 册　定价:98.00 元

前　言

O dear Ophelia!
I am ill at these numbers:
I have not art to reckon my groans.
亲爱的奥菲利娅;
这些数真让人烦恼:
*我可没有计算我的愁怀的技巧。**
— Hamlet (Act II, Scene 2, Line 120)

本书所讨论的算法直接地涉及数。但我相信把它们叫做*半数值算法*是适当的，因为它们处于数值和符号计算的边界线上。每个算法不仅计算数值问题所要求的答案，而且也应与一台数字计算机的内部操作很好地融合。在许多情况下，人们都不可能充分品味某个算法的美，除非他也懂计算机机器语言；相应的机器程序的有效性是不能同算法本身分开的一个重要因素。问题是寻找出计算机处理数的最佳方法，这既要考虑数值，又要研究策略。因此本书的主题显然既是数值数学的一个部分，也是计算机科学的一个部分。

在数值分析的"高层次"上工作的某些人将把这里处理的课题当做是系统程序员的领域，而工作在系统程序设计"高层次"上的其他人将把这里处理的课题当做数值分析的领域。但我相信，还会剩下一些人，他们将愿意仔细地考察这些基本方法。尽管这些方法或许处于低层次上，但是它们却奠定了计算机在数值问题上的所有更强大应用的基础，因此了解它们就很重要了。在这里我们最关心的是数值数学和计算机程序设计之间的界面。正是这两种技巧的结合，才使这一课题变得如此有趣。

本书比起本丛书的其他卷来，其数学内容所占比例要显著地高得多。这是由于所处理的课题所致。大多数情况下，在这里所展开的必要的数学课题几乎都是从很皮毛的内容开始的(或者从第 1 卷证明的结果开始的)。但是在若干部分，显然仍需要读者具有一定的微积分学知识。

本卷是由整套丛书的第 3 章和第 4 章组成的。第 3 章涉及"随机数"：它不单单是对生成随机序列的各种方法的研究，它还研究随机性的统计检验，以及一致随机数与其它类型随机量间的转换；后一课题说明在实践中如何使用随机数。本章还有

---

* 朱生豪译文为："亲爱的奥菲利娅啊，我的诗写得太坏；我不会用诗句来抒写我的愁怀。"(《哈姆莱特》，朱生豪译，吴兴华校，1997 年人民出版社出版。)鉴于译文难以准确表达作者的意旨，本书特将作者引文原文附上。以下同。——本书责任编辑

一节包括了随机数本身的性质。第4章的意图是讲述经历了数百年的进步之后，人类对算术运算的有趣发现；它论述了表示数的各种系统，以及在这些系统之间如何进行转换；而且它还处理关于浮点数、高精度整数、有理数、多项式及幂级数的算术运算，也包括因子分解和求最大公因子等问题在内。

第3章和第4章均可作为从大学三年级到研究生层次的一学期课程的基础。尽管"随机数"和"算术"的课程现在都不是许多大学课程表的一部分，但我相信，读者会发现这两章的学科内容本身是有实际教育价值的，非常适合于统一论述。我本人的经验是，这些课程是向大学生们介绍初等概率论和数论的很好的手段。通常，在这样的入门性课程中讨论的几乎所有课题都很自然地在同应用相关联中出现，而这些应用可以成为促进学生学习和鉴赏理论的重要因素。其次，每一章都给出一些更深入课题的提示，它们将激发许多学生进行进一步研究的兴趣。

本书的大部分内容都是自成体系的，除了偶尔涉及在第1卷中说明的MIX计算机的讨论外。附录B列出了本书所用的数学符号，其中一些符号与传统数学书中略有差别。

第3版前言

本书的第2版完成于1980年，它实际上也是电子排版系统 T_EX 和 METAFONT 的第一个重大的测试实例。现在我高兴地用她（正是为了她我萌发了开发电子排版系统的念头）的第3版庆祝 T_EX 和 METAFONT 的全面开发成功。最终我将能以一致的格式拥有《计算机程序设计艺术》的所有各卷，而这将使它们很容易适应未来在印刷和显示技术上的变化。这些进展已经使我得以把长期以来一直想要做的数以千计的改进收到这些书中。

我逐字逐句地审阅了新版的所有文字，在或许加上一些更为深思熟虑的论述的同时，我试图保留原来句子的朝气；增添了几十道新习题，还对几十道旧的习题给出了新的和改进了的答案。变动随处可见，但最主要的是在3.5节（关于随机性的理论保证），3.6节（关于可移植的随机数生成程序），4.5.2小节（关于二进制的最大公因子算法，以及4.7节（关于幂级数的合成和迭代）。


然而，《计算机程序设计艺术》的写作仍然是进行中的工作。关于半数值算法的研究继续以非凡的速度在发展着。因此，本书的某些部分被冠以"正在施工"的图标，用于对该部分的内容还不是最新表示歉意。我的文件中已经挤满了许多重要的素材。我打算从现在开始用大约16年的时间，把这些素材包含在第2卷最后的辉煌的第4版中，但是，我必须首先完成第4卷和第5卷，而且除非绝对需要，我一

刻也不想拖延它们的出版。

我要向在过去35年来帮助我搜集和改进这些素材的数百位人们致以最衷心的感谢。准备新版本的大多数困难工作是由Silvo Levy完成的，他熟练地编辑电子文本，而Jeffrey Oldham则负责把几乎全部图形转换成METAFONT的格式。我已经把机敏的读者们在第2版中所发现的每一个错误(以及，竟无人发现的一些错误也在内)都作了改正；而且我已经力图在新的材料中不引进新的错误。然而，我想仍然会存在某些缺点，我要尽快地改进它们。因此我将高兴地支付2.56美元给每一个技术、印刷或历史错误的头一位发现者，在网页 http://www-cs-faculty.stanford.edu/~knuth/taocp.html 列出了向我报告过的所有错误的更正。*

Stanford, California D.E.K.

July 1997

When a book has been eight years in the making,
there are too many colleagues, typists, students, teachers, and friends to thank.
Besides, I have no intention of giving such people
the usual exoneration from responsibility for errors which remain.
They should have corrected me!
And sometimes they are even responsible for ideas
which may turn out in the long run to be wrong.
Anyway, to such fellow explorers, my thanks.

当一本书已经编写了8年时，需要感谢的
同事、打字员、学生、老师和朋友真太多了。
此外，我无意对这些人宣布，对于遗留的错误，他们不负责任。
他们应当帮助我改正它们！
而且有时他们也应对一些经过长时间考验被证实是错的那些思想负责。
但无论如何，对于这样的同行探索者，我都要表示衷心感谢！

— EDWARD F. CAMPBELL, JR. (1975)

'Defendit numerus,' [there is safety in numbers]
is the maxim of the foolish;
'Deperdit numerus,' [there is ruin in numbers]
of the wise.

"数能保平安"是极大的愚蠢；
"数中有祸"是明智的。

— C. C. COLTON (1820)

* 中文版已按2001年11月的最新勘误表更正了原书的错误。网页 http://www.ndip.com.cn/computer 上也将列出对中文版的勘误表，我将很诚挚地对发现中文版错误的读者表示感谢，但我不敢保证也支付您2.56美元。——本书责任编辑

关于习题的说明

本套书的习题既可用于自学，也可用于课堂练习。无论是谁，如果想纯粹地通过阅读，而不将所阅读的信息应用到特定问题上，并由此牵引思考先前阅读的内容，就想学到一门学问，纵然可能，那也是很困难的。其次，对于我们自己的发现，我们总是领会得最透。因此，习题形成了这一套书的一个重要部分；我着意使这些习题含有丰富的信息，而且也尽量选择既有趣又有启发性的习题。

在许多书里，容易的习题被随机地混杂于极端困难的问题当中。有时这是不合适的，因为读者预先想要知道一道习题花多少时间——否则他们可能跳过这些习题了事。这一情况的典型例子是由 Richard Bellman 所著的*Dynamic Programming* 一书。这是一本重要的先驱性的论著，其中在某些章的末尾，在"习题和研究题"的标题下，把一组问题收集在一起，而且一些极其平凡的问题出现在一些深入的、还未被解决的问题当中。据说，有人曾问过 Bellman 博士，怎样区分习题和研究题。他回答说："如果你能解决它，那它就是一道习题，否则它就是一个研究题。"

但在这一类书里把研究问题和很容易的习题都包括进来确有其道理；因此，为使读者免于陷入确定哪是习题哪是研究题的困境，我给习题注明了等级分，以指出困难的程度。这些分数大体具有下列意义：

分数	解释
00	一个极其容易的问题，如果你已理解正文的内容，就可立即做出回答。这样一道题几乎总是可以"眉头一皱"就把它做出。
10	一个简单的问题。它要求你去思考刚刚学过的内容，但绝不意味着是困难的。你应当有能力在顶多一分钟之内就把它做出。在获得解答的过程中可能要用到笔和纸。
20	一个普通的问题。它检查你对正文内容的基本理解，但你可能需要十五或二十分钟才能完整地回答它。
30	一个中等难度或中等复杂的问题。这个题目可能需要两个小时以上的工作才能令人满意地解决。或者甚至更长时间，如果电视机在开着的话。
40	确实是一个十分困难或冗长的问题。在学校里，它将适合于作为一个学期的课程设计。一个学生应当有能力在相当长的时间里解决它，但这个解不会是平凡的。
50	就作者在编写本书时所知，这是一个还未令人满意地解决的问题，尽管已经有很多人做了尝试。如果你已经找到了这样一个问题的答案，你应当把它写出来发表。其次，本书的作者将乐意尽快地听到关

于这一解答的消息(当然,假定它是正确的)。

通过在这个"对数"尺上的内插,其它分数的意义也就清楚了。例如 17 分将表示比普通的题更为简单的一道习题。一个随后被某个读者解决了的 50 分的题在本书后来的版本中将以 45 的分数出现,而且会在互联网上的勘误表中刊出(参见版权页)。

分数除以 5 的余数表示所要求的详细工作量。因此分数是 24 的习题可能要比分数是 25 的题花费更长的时间来求解,但后者将要求更多的创造性。

作者已经认真地试图指定精确的分数,但是提出问题的人要想确切地知道,所提问题对求解的另一个人难度如何,肯定是困难的;而且每个人都会有较其他人更为适应的问题类型。只能希望这些分数较好地反映了习题的困难程度,希望读者把它们当做一般的导引,而不应作为绝对的指标。

本书是为具有不同程度的数学训练和素养的读者写的;有些习题是特意为有更多数学基础的读者提供的。如果一道题的出题动机或涉及的数学概念超越了主要兴趣仅仅是算法编程本身的读者应掌握的程度,则在分数前边加上 M。如果一道习题的答案必须涉及在本书中未予提供的微积分或其它高等数学知识,则对这道题标以"*HM*"的字母。"*HM*"标记并不必定意味着困难。

某些习题的前边标有三角符号"▶";这说明是特别有启发性的问题,因而特别予以推荐。当然,并不期望读者/学生来求解所有的习题,所以标出了那些看起来最有价值的部分(这并不意味着贬低其它习题!)。每个读者至少应该尝试去解分数是 10 或者更低的所有问题;三角符号可以帮助指出应该对具有较高分数的哪些问题予以优先考虑。

在答案部分中已经列出大多数习题的解答,请明智地使用它们。在你已真正地做出努力亲自解决问题之前,不要去翻答案,除非你确实没有时间来做这一特定的习题。在获得了你自己的解答或者对该题做了郑重的尝试之后,你可能会感到答案是有启发和有帮助的。给出的解答通常十分简短,作者认为你已经认真地通过自己的方法做过求解尝试,因而略去了其细节。有时解答所提供的信息比所要求的为少,但通常都是更多的。十分可能,你有比这里给的更好的答案,或者在答案中你发现一个错误。在这样的情况下,作者将乐意知道详细的情况。在本书以后的版本中将在适当地方刊登出这些改进了的答案以及提供这些解的人的姓名。

当做一道习题时,一般地你可以使用前边习题的答案,除非明确地禁止这样做。在对习题打分时,已经考虑到这一点了,因此很可能第 $n+1$ 题的分数比第 n 题还要低,尽管它把第 n 题的结果作为一个特殊情况包括进来。

代码含义			
		00	立即可解的
		10	简单的(一分钟)
		20	一般水平(一刻钟)
▶	特别推荐的	30	难度适中
M	面向数学的	40	学期设计
HM	要求"高等数学"的	50	研究题

习　题

▶1.[*00*] 分数"M20"意味着什么?

2.[*10*] 在一本教科书中的习题对读者来说有什么价值?

3.[*34*] Leonhard Euler 在 1772 年猜测,方程 $w^4+x^4+y^4=z^4$ 没有正整数解,但 Noam Elkies 在 1987 年证明,它有无穷多个解[见*Math. Comp.* **51** (1988),825～835]。试求使得 $0\leqslant w\leqslant x\leqslant y<z<10^6$ 的所有整数解。

4.[*M50*] 试证明,当 n 是整数,$n>4$ 时,方程 $w^n+x^n+y^n=z^n$ 没有正整数的 w,x,y,z 的解。

Exercise is the beste instrument in learnyng.
习题是学习的最好手段。
ROBERT RECORDE, *The Whetstone of Witte* (1557)

内 容 简 介

本书是国内外业界广泛关注的7卷本《计算机程序设计艺术》第2卷的最新版。本卷对半数值算法领域做了全面介绍，分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论，广泛剖析了计算机程序设计与数值分析间的相互联系，其中特别值得注意的是作者对随机数生成程序的重新处理和对形式幂级数计算的讨论。

本书附有大量习题和答案，标明了难易程度及数学概念的使用。

本书内容精辟，语言流畅，引人入胜，可供从事计算机科学、计算数学、计算技术诸方面的工作人员参考、研究和借鉴，也是相关专业高等院校的理想教材和教学参考书。

目　录

第3章　随　机　数

第4章　算　　术

第3章 随 机 数

Any one who considers arithmetical methods
of producing random digits is, of course, in a state of sin.
任何考虑用算术方法产生随机数字的人
都是在做错事。
— JOHN VON NEUMANN (1951)

Lest men suspect your tale untrue,
Keep probability in view.
为了避免人们怀疑你的故事的真实性,
请不要忘了概率。
— JOHN GAY (1727)

There wanted not some beams of light
to guide men in the exercise of their Stocastick faculty.
不需要什么光芒来照引人们
去使用他们的随机本领。
— JOHN OWEN (1662)

3.1 引 言

"随机地选择"的数,有许多不同类型的应用。例如:

a)*仿真* 当使用一台计算机仿真自然现象时,就需要用随机数使事情变得逼真。仿真涉及从核物理(其中粒子受到随机的碰撞)直到运筹学(比如说,人们在随机的时间里进入一个飞机场)的许多领域。

b)*抽样* 通常,要考察所有可能的情况是不实际的,而随机的抽样将使我们能了解"典型"的行为。

c)*数值分析* 利用随机数已经想出了许多巧妙的技术来解复杂的数值问题。关于这个课题,已有不少专著。

d)*计算机程序设计* 为检验计算机算法的有效性,随机值乃是数据的好来源。更重要的是,随机值对于随机化算法的运算是很要紧的,随机化算法经常比确定性算法要优越得多。随机数的使用是我们在这套书中所关注的主要应用,所以我们要在第3章,即在讲其它大多数计算机算法之前,先讲随机数。

e)决策　据说,有许多决策人通过抛硬币或掷飞镖等等来做出他们的判断。还听说某些大学教授也以此为基础来评分。有时,做出这样完全"无偏见"的判断是重要的。随机性也是矩阵游戏理论中优化策略的必不可少的部分。

f)美学　一点儿随机性就会使计算机生成的图形和音乐似乎更活泼。例如,下面左边的图案在某些场合就比右边的更有吸引力。

[请见 D. E. Knuth, *Bull. Amer. Math. Soc.* **1** (1979),369。]

g)娱乐　摇骰子、洗扑克牌、转轮盘等,是人们很喜爱的娱乐方式。随机数的这些传统用法,已经被命名为"蒙特卡罗方法",它已成为描述任何使用随机数的算法的通用术语。

考虑这一课题的人们几乎总是碰到关于"随机"一词含义的哲学性讨论。在某种意义下不存在什么随机数。例如,2 是一个随机数吗?我们宁愿谈论具有一个确定分布的独立的随机数序列。这大致意味着,每个数的出现都只是偶然的,并且与这序列中的其它数无关,每个数落入任何给定的范围都有一个确定的概率。

在一个有限数集上的一个一致分布是一个这样的分布,其中每个可能的数都有相等的概率。凡分布一般都认为是一致的,除非特别说明是某种其它分布。

在一个(一致的)随机数字序列中,0 到 9 这 10 个数中每一个数出现的次数约为 1/10,每一对连续数字出现的次数约为 1/100,等等。然而,如果我们取一个有 100 万个数字的真正的随机序列,则它总不会恰好有 10 万个 0,10 万个 1,等等。事实上,这种机会是十分微小的;而由这种序列组成的序列,则平均地具有这一特征。

100 万个数字的任何一个特定的序列,和任何其它序列都是同等可能的。因此,如果我们正随机地选择 100 万个数字,而其中的头 999 999 个碰巧已经是零,则最后一个数字是零的机会,在一个真正随机的状态下,仍然恰恰是 1/10。这些说法对很多人来说似乎是自相矛盾的,但事实上并非如此。

对于一个随机序列的抽象定义,有许多好的描述方式。我们将在 3.5 节回头来讨论这个有趣的问题。但眼前,还是让我们来直观地了解这一概念吧!

许多年前,那些在科学工作中需要随机数的人们,用从一个"充分转动起来"的坛子中抓出球来或摇骰子、分牌之类的办法获得随机数。1927 年,L. H. C. Tippett 发布了超过 40 000 个随机数字的一张表,那些数字是"从人口统计调查报告中随机地取得的"。自那以后,已经造出了一些特殊的装置,用来机械地生成随机数。1939 年,M. G. Kendall 和 B. Babington-Smith 使用头一部这样的机器造出了有 100 000 个随机数字的一张表。1951 年首次安装的 Ferranti Mark I 计算机有一个内置指令,它使用一个电阻噪声生成器将 20 个随机位放进累加器中;这一功能受到了 A. M. Turing 的推荐。1955 年,RAND 公司又公布了一张被广泛使用的有百万个随机数字的表,这张表是利用另一部特殊装置得到的。一台取名为 ERNIE(厄尼)的著名随机数机器被使用了许多年,用于产生英国政府有奖债券的中奖号码。[见 Kendall 和

Babington-Smith 的论文,*J. Royal Stat. Soc.* **A101** (1938),147～166;**B6** (1939),51～61。也见 S. H. Lavington 对 Mark I 的讨论,*CACM* **21** (1978),4～12;以及对 RAND 表的评论:*Math. Comp.* **10** (1956),39～43。也见 W. E. Thomson 对 ERNIE 的讨论,*J. Royal Stat. Soc.* **A122** (1959),301～333。]

在计算机问世之后不久,人们开始探求在计算机程序中求随机数的有效方法。可以使用一张表,但由于内存空间和输入时间的要求,这种方法的实用性有限,因为这种表可能太短了,而且编制和维护这种表也是一件麻烦的事情。可以把诸如 ERNIE 这样的机器联到计算机上,如同在 Ferranti Mark I 中所做的那样,但这也并不令人满意,因为当校验一个程序时,它实际上不可能再一次精确地重复以前的计算;而且,这类机器还经常会发生不易查出的故障。20 世纪 90 年代技术的发展又使表变得有用起来,因为 CDROM 上可以分布(存储)十亿字节的经过测试的随机数。1995 年,George Marsaglia 将一个噪声二极管的电流输出与确定的经过扰频的 rap 音乐叠加在一起生成了 650MB 的随机值(他称之为黑白噪声)并把它们做成演示光盘,因而促进了随机数表的复兴。

早期用机械方式生成随机数的缺点,引起了人们利用一台计算机的普通算术操作来产生随机数的兴趣。1946 年前后,John von Neumann(冯诺伊曼)头一个建议利用这种方法;他的办法是取前面的随机数的平方,并抽取中部的数字。例如,如果正在生成 10 位数字,而且先前的值是 5772156649,则把它平方得到

3331772380594909201

因此下一个数就是 7923805949。

对于这种技术,有十分明显的异议:既然每个数完全由它先前的数所决定,那么以这样的方法产生的序列怎么会是随机的呢?(请见本章开始处冯诺伊曼的评述。)回答是,这个序列不是随机的,仅仅像是随机的而已。在典型的应用中,一个数与跟在它后面的那个数之间的真实关系并无客观的意义,因此,这种非随机的特征并不真是不可取的。从直观上看,平方取中似乎相当充分地搅乱了前面的数。

用这样一种确定的方法生成的序列,在高深的学术著作中通常都叫做伪随机或拟随机序列,但在本书中我们简单地称之为随机序列,只不过把它们理解成只是看起来像随机的罢了。也许任何随机序列最多只能说是"显然随机的"。凡是仔细地选择了适当的方法的,在计算机上以确定方式生成的随机数,都已十分成功地用于几乎每一项应用之中。当然,确定的序列并不总是答案,它们肯定不应代替 ERNIE 来抽奖。

已经证明,冯诺伊曼原来的"平方取中法"并不是求随机数的好方法。危险在于这个序列容易出现重复元素的短循环。例如,如果 0 一旦作为这个序列的一个数出现,则它将不断地重现本身。

一些人在 20 世纪 50 年代初曾以"平方取中法"进行了实验。G. E. Forsythe 用 4 位数字代替 10 位数字,试验了 16 个不同的初始值。结果发现,其中的 12 个导致了以循环 6100,2100,4100,8100,6100,… 为结局的序列,而其中有两个退化成零。N. Metropolis对平方取中法进行了广泛的试验,试验中大多采用二进数系统。他证

明了:用 20 位数字进行工作时,序列可能退化成 13 个不同的循环,其中最长的循环周期为 142。

当发现了 0 时,很容易用一个新的值重新开始平方取中法,但不容易避免长的循环。习题 6 和 7 讨论了确定循环序列周期的某些有趣方法,只须使用很少的存储空间。

在习题 9 和 10 中,指出了平方取中法的一个理论上的缺陷。另一方面,N. Metropolis用 38 位数工作,在出现退化之前得到了大约 750 000 个数的序列,而且得到的 750 000×38 位数字满意地通过了随机性的统计检验。[*Symp. on Monte Carlo Methods* (Wiley,1956),29～36。]这个实验证明,平方取中法能给出有用的结果,但是在没有实施精心的计算之前,对它过于信任还是危险的。

在最初编写本章时,人们所使用的许多随机数生成程序都不是很好的。人们习惯于不去研究这样的子程序,某些相当不令人满意的老方法仍被盲目地从一个程序员传到另一个程序员,以至用户对它原有的局限性一无所知。在本章我们将看到,掌握有关随机数生成程序的要点并不困难,尽管为避免通常易犯的错误,谨慎从事是必要的。

要想出一个十分简单明了的随机数生成程序并不容易。作者在 1959 年就对此有深刻体会,当时试图利用下列特定的方法来建立一个非常好的随机数生成程序:

算法 K("超随机"数生成程序)　给定一个 10 位的十进制数 X,本算法可以用来把 X 变为一个像是随机序列中的下一个数。尽管可以预期这个算法能得出一个相当随机的序列,但后面提到的理由说明了它事实上并非十全十美。(读者除了注意这个算法的复杂性外,不必详尽地研究它。请特别注意 K1 和 K2。)

K1. [选择迭代次数]　置 $Y \leftarrow \lfloor X/10^9 \rfloor$,即置 X 的最高位有效数字。(我们将执行步骤 K2 到 K13 $Y+1$ 次;亦即,我们应用随机变换随机次。)

K2. [选择随机步骤]　置 $Z \leftarrow \lfloor X/10^8 \rfloor \bmod 10$,亦即,置 X 的第二位最高有效数字,转向步骤 K(3 + Z)。(亦即,我们转到程序中的一个随机步骤。)

K3. [确保$\geq 5\times 10^9$]　如果 $X < 5000000000$,则置 $X \leftarrow X + 5000000000$。

K4. [平方取中]　以 $\lfloor X^2/10^5 \rfloor \bmod 10^{10}$ 代替 X,亦即,以 X 平方取中来代替 X。

K5. [乘]　以 $(1001001001X) \bmod 10^{10}$ 代替 X。

K6. [伪补数]　如果 $X < 100000000$,则置 $X \leftarrow X + 9814055677$;否则置 $X \leftarrow 10^{10} - X$。

K7. [互换两半]　把 X 的后五位数字与前五位数字互换,即置 $X \leftarrow 10^5(X \bmod 10^5) + \lfloor X/10^5 \rfloor$,即 $(10^{10}+1)X$ 的中间 10 位数字。

K8. [乘]　同步骤 K5。

K9. [减小数字]　把 X 的十进表示的每个非 0 数字减 1。

K10. [99999 修改]　如果 $X < 10^5$,则置 $X \leftarrow X^2 + 99999$,否则置 $X \leftarrow X - 99999$。

K8. [乘]　同步骤 K5。

K9.[减小数字] 把 X 的十进表示的每个非 0 数字减 1。

K10.[99999 修改] 如果 $X<10^5$,则置 $X\leftarrow X^2+99999$,否则置 $X\leftarrow X-99999$。

K11.[规格化] (这时 X 不能为 0。)如果 $X<10^9$,则置 $X\leftarrow 10X$ 并重复这一步骤。

K12.[修改过的平方取中法] 以 $\lfloor X(X-1)/10^5\rfloor \bmod 10^{10}$ 代替 X,亦即,以 $X(X-1)$ 的中间 10 位数字代替 X。

K13.[重复?] 如果 $Y>0$,则 Y 减 1 并返回到步骤 K2。如果 $Y=0$,则本算法终止,以 X 作为所求的"随机"值。 ▌

(对应于上述算法的机器语言程序竟是如此复杂,以至不借助于注解来读它的清单时,人们将不知道这个程序是干什么的。)

考虑到算法 K 的复杂设计,这一算法似乎能产生无穷尽的令人难以置信的随机数,但事实上,当把这个算法头一次放到计算机上时,它几乎立即收敛到 10 位数值 6065038420——非常巧合,本算法把这个数转换成它自己(见表 1)。若以另外一个数开始,则这个序列在 7 401 个值之后,开始以长度为 3 178 的周期进行循环。

表 1 非常巧合:算法 K 把数 6065038420 变成它自己

步骤	X(之后)		步骤	X(之后)	
K1	6065038420		K9	1107855700	
K3	6065038420		K10	1107755701	
K4	6910360760		K11	1107755701	
K5	8031120760		K12	1226919902	$Y=3$
K6	1968879240		K5	0048821902	
K7	7924019688		K6	9862877579	
K8	9631707688		K7	7757998628	
K9	8520606577		K8	2384626628	
K10	8520506578		K9	1273515517	
K11	8520506578		K10	1273415518	
K12	0323372207	$Y=6$	K11	1273415518	
K6	9676627793		K12	5870802097	$Y=2$
K7	2779396766		K11	5870802097	
K8	4942162766		K12	3172562687	$Y=1$
K9	3831051655		K4	1540029446	
K10	3830951656		K5	7015475446	
K11	3830951656		K6	2984524554	
K12	1905867781	$Y=5$	K7	2455429845	
K12	3319967479	$Y=4$	K8	2730274845	
K6	6680032521		K9	1620163734	
K7	3252166800		K10	1620063735	
K8	2218966800		K11	1620063735	
			K12	6065038420	$Y=0$