算法设计与分析

姜新文 彭立宏 殷建平/ 编著

SUANFA SHEJI YU FENXI

SUANFA
SHEJI YU FENXI

国防科技大学出版社

算法设计与分析

姜新文　彭立宏　殷建平　编著

国防科技大学出版社
·长沙·

图书在版编目(CIP)数据

算法设计与分析/姜新文,彭立宏,殷建平编著.—长沙:国防科技大学出版社,2008.9
ISBN 978-7-81099-495-8

Ⅰ.算… Ⅱ.①姜… ②彭… ③殷… Ⅲ.①电子计算机—算法设计 ②电子计算机—算法分析 Ⅳ.TP301.6

中国版本图书馆 CIP 数据核字(2008)第 060367 号

国防科技大学出版社出版发行
电话:(0731)4572640 邮政编码:410073
http://www.gfkdcbs.com
责任编辑:唐卫葳 责任校对:文 慧
新华书店总店北京发行所经销
国防科技大学印刷厂印装
*
开本:787×1092 1/16 印张:13.25 字数:314 千
2008 年 9 月第 1 版第 1 次印刷 印数:1-5000 册
ISBN 978-7-81099-495-8
定价:22.80 元

前言

本书是作者在多年讲授《算法设计与分析》课程基础上编写的，适应本科生、研究生等层次的教学对象，适应将《数据结构》与《算法设计与分析》分开讲授和学习的教学目标。作者多年的教学实践证明，如果学习方法得当，掌握了一定的技巧，参加程序设计竞赛的优秀高中学生通过阅读本书，可以获得智力启发，取得不俗的成绩。

《算法设计与分析》的教学内容的组织有两种思路。一种思路是围绕问题或者问题类展开，将用不同方法求解同一个问题的算法集中。这种组织方式的好处是便于把握问题求解研究过程的发展脉络，同时集中介绍的算法可以形成手册。另一种思路是围绕算法设计的方法、策略展开，将用同样的方法求解不同问题的算法集中。这种组织方式的好处是便于体会、掌握算法设计的方法。本书采用第二种思路组织教学内容。但是，由于本书在编写过程中尽量将同一个问题的不同求解算法贯穿始终，因此读者仍然可以通过分布在不同章节的、求解同一个问题的不同算法的比较学习，了解问题求解研究过程的发展脉络。

全书共分七章。第一章介绍基本概念和数学基础。第二章、第三章、第四章、第五章、第六章分别介绍分治法、动态规划法、贪心法、回溯法、分枝限界法的基本思想和用相关方法设计的算法实例。第七章介绍 NP 完全问题。基于科学和技术应该解决现实世界存在的问题、应该面对未来世界将出现的问题的这种认识，本书讨论的算法设计实例按照现实有用的原则选取，并将随着时代进步、技术更新淘汰出局的问题舍弃。

在此要感谢众多参考书目的作者，同样的内容通过不同的叙述，体现独到的理解和独运的匠心，通过阅读他们的书我们得到丰富的养分；还要感谢互联网上的众多作者，因为本书有些内容来自网上。虽然网上有些东西不严格、不系统，存在错误，但仍然是对某些问题认识的智慧表达。由于时间仓

促，水平所限，书中定有粗疏错误，我们热忱欢迎对本书提出批评和指正。

算法的学习是有难度的，怎样学习好算法的问题经常被问到。我们曾经读过两位赫赫有名的算法大家的书。一本书在扉页上标明“本书所有习题，做得出来的是习题，做不出来的是研究题”。另一本书写道，真正掌握一种知识有两个最好的途径，一是教会一个人，二是教会计算机。总结他们的话，我们认为掌握算法的途径有三点：一是思考，二是讨论，三是实现。我们知道，集思考、讨论、实践于一体的思考，是能够将思考引向深入的思考，是能够产生新认识的思考。多思是解罔的唯一之道。

其实算法不仅仅是计算机专业应该考虑的事情。世界上任何问题都有一定的解决步骤，而算法研究的就是求解问题的步骤。一个算法修养水平高的人，有条件成为好的问题简化专家，有条件成为有条理和有效率的事务处理者，有条件成为面对复杂纷繁的局面洞若观火、冷静精明的组织管理者。2008 年北京奥运会的组织安排不就是一个“算法”吗？是一个伟大民族精心设计的“算法”最精彩的演绎！只是，这一次，算法的执行者不是计算机，而换成了人。愿算法像数学一样成为普世的修养。

2008 年 8 月 7 日
于科大佳园

目　录

第1章 算法复杂性及其分析

1.1 概 述

用计算机求解问题一般要经历以下过程：

(1)描述问题。为了求解问题，首先必须了解问题的实质，然后将所有已知条件以及所需要的答案描述清楚。

(2)数学建模。数学建模实际上就是给出需要求解的问题的数学定义或者数学描述。对于问题求解而言，这是非常重要的工作，具有很大的创造性。没有万能的方法可以指导建模过程，它取决于研究者的知识结构和工作经验。

(3)算法设计与分析证明。模型一旦建立，就可以进行算法设计。算法设计同样是具有创造性和挑战性的工作，没有适合所有问题的万能方法。研究者需要充分发挥创造性，充分运用已有知识和抽象思维能力，慢慢形成解决问题的思路，勾勒出具体的求解步骤。算法设计一旦完成，就需要对算法进行分析证明，为确认算法能够实现求解目标的必然性提供数学和逻辑的依据。算法设计与分析证明是一个反复的过程。分析证明会发现设计的漏洞或者发现改进的思路，于是形成修改的设计，然后对修改的设计进行分析证明。

(4)实现和验证。将给定的算法正确地转换成一个程序，得到运行结果，并判断结果是否与预期一致。

本书只讨论算法的设计与分析。

非形式地说，算法是一个有穷规则的有序集合。这些规则确定了解决某一类问题的一个运算序列，对于该类问题的任何初始输入，它能机械地、一步一步地计算，并在有限步后终止计算，产生输出。

算法有如下五大特征：

(1)有穷性。一个算法包含的规则条数是有穷的。执行过程必须在有穷个计算步后终止。

(2)确定性。算法中给出的每一个规则，必须是精确定义的、无歧义的。

(3)能行性。对每个计算步而言，必须能够在有限的时间内完成。

(4)输入。有零个或者多个输入信息。

(5)输出。至少产生一个输出信息，这些输出信息通常被解释为“对于输入的计算结果”。

可以通过以下问题求解的例子来说明算法的特性。给定两个正整数 m 和 n，求它们

的最大公因子(m, n)。

这个问题可以采用辗转相除算法求解。这个方法在西方被称作欧几里德算法。中国古代数学家秦九韶在《九章算术》一书中记载了这个算法。这个算法可以用三个语句描述:

(1)以 n 除 m 得余数 r, $0 \leqslant r < n$;

(2)如果 $r=0$, 输出 n 的当前值, 算法结束, 否则继续执行第(3)步;

(3)将 n 的当前值送 m, r 的当前值送 n, 即执行 $m \leftarrow n$, $n \leftarrow r$, 然后转第(1)步。

上面的辗转相除算法规定了三个计算步骤, 其中每个计算步骤的意义都是明确的、能行的。虽然算法需要循环, 但是, 对任意给定的 m 和 n, 由于 r 在计算过程中单调下降, 所以有限步计算之后, 必然出现 $r=0$ 的情况, 从而导致算法终止并产生输出。所以, 上面给出的求 m 和 n 的最大公因子的三个步骤就是一个算法。

人们关注算法、研究算法的第一个原因是问题求解的需要。算法是解决问题的思路和具体的求解步骤。有了算法才能够编制程序并最终解决问题。显然, 如果不能找到求 m 和 n 的最大公因子的算法, 即使对程序设计语言再熟悉, 也不会知道如何编程并用计算机求解, 或者个人算术运算的基本技能再强, 也不会知道如何计算出最大公因子。

人们关注算法、研究算法的第二个原因是求解效能的需要。

虽然还没有严格定义算法的复杂性并阐述如何具体分析它, 但是可以先用能够理解的方式认识复杂性对问题求解的影响。算法设计就是寻找问题求解的步骤, 这些步骤当然由一些"动作"构成, 完成一个"动作"总是需要一定时间的, 所有"动作"耗费的时间总和构成算法耗费的总时间。由于不同个体执行同一"动作"的时间是有差异的, 所以我们用一个算法中"动作"的个数作为该算法耗费的时间的计量。自然, 求解同一个问题的好的算法应该减少"动作"的个数。

《水浒传》中有个神行太保叫做戴宗, 可以日行千里。如果要送一封千里邮程的信, 戴宗只需要一天, 而常人需要十日。如果现在梁山有一批信件需要送抵许多地方, 宋江选择戴宗完成任务就涉及到类似算法设计以及计算机选择的全部工作:

宋江制定信件的送抵顺序类似于算法设计。有的顺序是错误的, 比如路不通, 等等; 有的顺序费时间一些, 比如路径重复, 等等。设计好的送抵顺序类似于设计正确的且省时间的算法。

宋江选择戴宗完成任务类似于我们选择一台计算机实现算法。选择戴宗类似于选择高速计算机。送抵顺序不变, 送信人能力越强, 完成时间越短; 送信人选择不变, 送抵顺序越优, 完成时间越短。这个过程相当于算法不变, 机器越快, 求解时间越短; 机器不变, 算法越优, 求解时间越短。

现在的问题是, 计算机运算速度如此之快, 还需要在意算法耗费的时间吗? 需要! 我们考察一个很熟悉的问题在计算机上求解的过程。要注意的是, 在这里, 计算机上的"动作"实际上就是运算和指令。

n 阶行列式计算是许多计算过程都要遇到的问题。我们在"线性代数"中已经学习过, 一个 n 阶行列式的值定义为该行列式的所有取自不同行不同列的元素乘积的代数和。即

$$|a_{ij}| = \sum_{j_1 j_2 \cdots j_n \text{的所有排列}} (-1)^{\tau(j_1 j_2 \cdots j_n)} a_{1j_1} a_{2j_2} \cdots a_{nj_n}$$

不难理解，按照定义，要计算一个行列式的值，需要计算 $n!$ 个代数项，每个代数项需要 $n-1$ 次乘法运算，于是共需要 $n! \times (n-1)$ 次乘法运算，实现 $n!$ 个代数项求和需要 $n!-1$ 次加法运算，这样加法和乘法运算总的次数为

$$n! \times (n-1) + n! - 1$$

假定有一个 100 阶行列式，有一台千万亿次计算机。说明一下，100 阶行列式计算问题在科学计算中属于小规模问题，千万亿次计算机在当前还是人类追求的目标。用这台千万亿次计算机求解该 100 阶行列式所需时间(换算成以年为单位)为：

$$\frac{\text{总的运算次数}}{\text{千万亿次计算机一年完成的运算次数}} = \frac{100! \times (100-1) + 100! - 1}{10^{15} \times 3600 \times 24 \times 365} \approx 2.96 \times 10^{137}(\text{年})$$

这已经远远超出了宇宙的寿命！

改用高斯消去法。“计算方法”课程中介绍，加法运算和乘法运算的次数都约为 $n^3/3$，于是总的运算次数约为 $2n^3/3$。同样，用这台千万亿次计算机求解该 100 阶行列式所需时间(换算成以秒为单位)为：

$$\frac{\text{总的运算次数}}{\text{千万亿次计算机一秒钟完成的运算次数}} = \frac{2 \times 100^3}{3 \times 10^{15}} \approx 0.67 \times 10^{-9}(\text{秒})$$

从按照定义计算行列式的值，到按照高斯消去法计算行列式的值，我们可以看到算法改进对于问题求解的至关重要的影响。

还可以进一步讨论算法效能对于问题求解的影响。

如同上面例子中所揭示的，算法耗费的时间同运算次数的多少有关，而运算次数的多少同问题的规模或大小(如上例中行列式的阶)有关，于是算法耗费时间可以记为 $T(n)$，$T(n)$定义成某种运算的个数，其中，n 为问题的规模。

设有 5 个算法 A_1，A_2，A_3，A_4，A_5，它们耗费的时间如表 1.1 所示。

表 1.1 5 个算法及其时间复杂性函数列表

算 法	时间耗费
A_1	n
A_2	$n\log_2 n$
A_3	n^2
A_4	n^3
A_5	2^n

假定一次运算需要一个单位时间，据表 1.1，算法 A_1 处理一个规模为 n 的问题需 n 个单位时间，其余类推。取时间单位为 1 毫秒，各个算法 1 秒钟、1 分钟、1 小时能够处理的问题规模如表 1.2 所示。

表 1.2 相同时间内不同算法能够处理的输入量

算法	时间耗费	1 秒钟能够处理的最大输入量	1 分钟能够处理的最大输入量	1 小时能够处理的最大输入量
A_1	n	1000	6×10^4	3.6×10^6
A_2	$n\log_2 n$	140	4893	2.0×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

从表 1.2 可以看出，时间耗费随 n 的增加快速增加的算法，在同样时间内处理问题的规模要小得多。如 A_5，1 小时只能处理一个规模为 21 的问题。很容易计算出来，即使是 1 年、100 年甚至 1000 年，A_5 也只能处理一个极小规模的问题，即使计算机运行速度提高 100 倍、10000 倍，A_5 能够处理的问题规模的增加也微乎其微。有人说，算法的改进可以使几十年来计算机性能改进的成就黯然失色，这种说法从某种意义上讲是有一定道理的。以前面行列式计算的问题为例，我们看到，如果不改进算法，无论计算机怎么先进，人类可能永远无法看到一个 100 阶行列式的值被计算出来的那一天。

算法研究的意义是明显的。

早在计算机产生之前，人们已经开始对算法进行研究。例如，中国古代对算盘以及珠算的研究都包含对计算理论和算法的贡献。20 世纪 40 年代以后，由于电子计算机的出现，带来大量理论和实际的计算问题，进一步推动了对算法的研究。算法成了一个无处不在的、频繁出现的术语，算法研究成为计算机科学和数学领域永恒和持续活跃的研究领域。

1.2 RAM 模型

为了能够给能行性和确定性一个精确的定义，为了度量“动作”耗费的时间，必须选择执行算法的计算模型。一个计算模型将对每一个基本的计算步有严格的定义。

下面介绍随机存取模型(Random Access Machine)，简称为 RAM。RAM 是一台单累加器计算机模型，这是一种很有用的抽象计算装置。它不允许程序修改其自身。一台 RAM 由以下部件组成：

(1)程序存储部件。一片特殊的存储器件，供存放程序(指令)。RAM 程序不能修改自身。

(2)累加器 R_0。其功能与一般计算机的累加器相同。

(3)内存储器。其存储单元依次命名为 R_1，R_2，R_3，…，总共有任意多个内存单元，

每个单元能存放一个任意大小的整数。这两个“任意”是 RAM 模型对于现实计算机的一种抽象与简化，它适用于这样的场合：

(i)求解问题所需要的单元不超过一台计算机的存储容量；

(ii)计算过程中出现的任何信息的大小不会超过计算机的字长。

(4)只读输入带。它由一系列方格和一个带头组成，每格可以存放一个任意大小的整数。每当机器从输入带上读出一个数时，带头(读出磁头)就自动向右移动一格。

(5)只写输出带。它也是由一系列方格和一个带头组成的。每格可以记录一个任意大小的整数。开始计算以前，所有方格全部为空白。一旦在输出带头下面的方格中打印了一个整数后，输出带头就自动右移一格。符号一经写出，不可再修改。

(6)程序控制部件。它主要包括指令计数器和操作码译码器等控制部件。它能控制一个程序按程序的走向和每条指令的严格定义一步步执行。

图 1.1 是随机存取模型 RAM 的总体结构示意图。

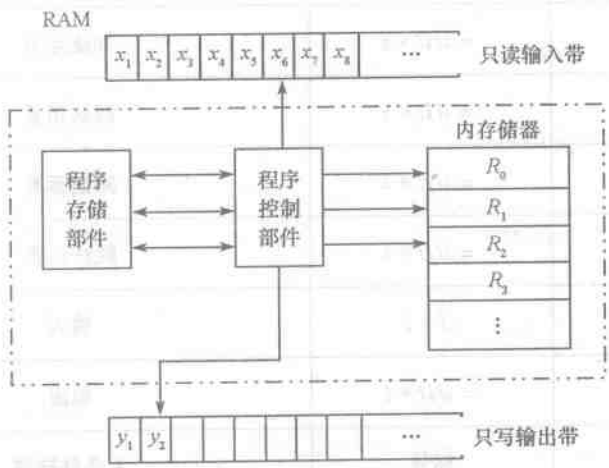

图 1.1　随机存取模型结构示意图

RAM 有算术运算指令、输入输出指令、数据存取指令以及转移指令等与实际计算机相同类型的指令，有直接寻址和间接寻址两种基本寻址方式。同通常的机器一样，所有的计算都在累加器 R_0 中进行，而 R_0 也可以容纳一个任意大小的整数。每条指令都由“操作码”和“操作数”两部分组成，其中操作数有以下三种形式：

(1) $=i$，称作直接数型。这是一条无地址指令，表示操作数是整数 i 本身。

(2) i，称作直接地址型。i 是一非负整数，表示操作数是内存单元 R_i 中的内容 $C(i)$。因为 R_0 和 $R_i(i\geqslant 1)$统一编址，所以 $C(0)$表示累加器 R_0 中的内容。

(3) $*i$，称作间接地址型。i 是一非负整数，表示操作数是内存单元 j 中的内容 $C(j)$，而 j 则是内存单元 i 中所存储的那个整数，即 $j=C(i)$，操作数是 $C(j)=C(C(i))$。当 $C(i)<0$ 时，$C(j)=C(C(i))$无定义。

如果记操作数 a 的值为 $V(a)$，则有

$$V(=i)=i$$
$$V(i)=C(i)$$
$$V(*i)=C(C(i))$$

显然，C 是从非负整数到整数的单射函数。

RAM 的基本指令集如表 1.3 所示。原则上，可以根据实际需要而增设其他指令。比如可以增加逻辑运算指令、字符操作指令或位操作指令等，以扩充指令系统的功能。

表 1.3 RAM 的指令系统

操作码	操作地址	说 明
LOAD	=i/i/*i	取一操作数至累加器中
STORE	i/*i	累加器中的数送入内存
ADD	=i/i/*i	加法运算
SUB	=i/i/*i	减法运算
MULT	=i/i/*i	乘法运算
DIV	=i/i/*i	除法运算
READ	i/*i	输入
WRITE	=i/i/*i	输出
JUMP	标号	无条件转移
JGTZ	标号	正转移
JZERO	标号	零转移
HALT		停机

表 1.4 列出了 RAM 每条指令的严格定义。其中，$C(0)$表示累加器 R_0 的内容。凡是没有定义的指令都是非法的。例如

STORE =i
READ =i

等指令，RAM 都无法执行。除数为零时也非法。RAM 对所有无定义的指令作停机处理。

表 1.4 RAM 指令的涵义，操作数 a 为 $=i/i/*i$

指 令	指 令 的 涵 义
LOAD a	$C(0)\leftarrow V(a)$
STORE i	$C(i)\leftarrow C(0)$
STORE $*i$	$C(C(i))\leftarrow C(0)$
ADD a	$C(0)\leftarrow C(0)+V(a)$
SUB a	$C(0)\leftarrow C(0)-V(a)$
MULT a	$C(0)\leftarrow C(0)\times V(a)$
DIV a	$C(0)\leftarrow\lfloor C(0)/V(a)\rfloor$*
READ i	$C(i)\leftarrow$当前输入字；带头右移一格
READ $*i$	$C(C(i))\leftarrow$当前输入字；带头右移一格
WRITE a	在只写输出带的当前方格(即带头所指的方格)上打印 $V(a)$后，带头右移一格
JUMP b	指令计数器$\leftarrow$标号 b 的值
JGTZ b	当 $C(0)>0$时，指令计数器$\leftarrow$标号 b 的值；否则，指令计数器$\leftarrow$指令计数器的当前值+1
JZERO b	当 $C(0)=0$时，指令计数器$\leftarrow$标号 b 的值；否则，指令计数器$\leftarrow$指令计数器的当前值+1
HALT 停机	

* 记号$\lfloor x\rfloor$表示取小于或等于 x 的最大整数，$\lceil x\rceil$表示取大于或等于 x 的最小整数。

RAM 在执行一个程序时的工作过程与计算机的工作过程基本相同。首先，指令计数器指向程序的第一条指令，输入带上已有一串输入数据(当这个程序要求输入某些数据时)，输入带头对准第一个输入字 x_1 所在的方格；输出带上全是空白，输出带头指向左端第一方格。当机器执行第 k 条指令时，如果被执行的指令不是转移指令和停机指令，机器就完成操作码定义的有关操作，然后指令计数器自动加 1，指向第 $k+1$ 条指令。当执行的指令是无条件转移指令，则无条件转向这条指令的操作地址部分的标号所指定的那条指令(设程序中的指令都是可以加标号的)。当执行的指令是条件转移指令，那么如果条件满足，则转向这条指令的操作地址部分的标号所指定的那条指令，如果条件不满足，则顺序执行下一条指令。当执行的指令是 HALT 时就停机。

机器执行输入指令时，就从只读输入带上读入一个数据。当机器执行输出指令时，就在只写输出带上打印结果。当然，一个程序也可能没有输入指令(或者它不需要输入数据；或者它需要的数据在程序中用直接数型指令给出)。一般说来，一个程序至少应有一条输出指令。

一般说来，一个 RAM 程序定义了一个从输入信息到输出信息的映射。可以对这种映射关系作不同的解释，但最重要的两种解释是：一个 RAM 程序或者计算一个函数，或者识别一个语言。

假定一个程序 P 总是从输入带上读入 n 个整数 x_1，x_2，…，x_n，并且在输出带的第一个方格上输出整数 y 后停机，我们就说程序 P 计算了函数 $f(x_1, x_2, \cdots, x_n)$，且得到函数值 $y=f(x_1, x_2, \cdots, x_n)$。

对 RAM 程序的另一种解释是把它当作一个语言接受器。一个语言 L 是一个有穷字母表 Σ 上的字符串的集合。设字母表的长度为 k，可以用整数 1，2，3，…，k 分别表示字母表 Σ 中的各符号。

将字符串 $S=a_1a_2\cdots a_n$ 放在输入带上，第 1 个方格放置 a_1，第 2 个方格放置 a_2，…，第 n 个方格放置 a_n。在输入带的第 $n+1$ 格放上数字 0，表示输入串的结束。如果一个 RAM 程序 P 在读入字符串 S 和结束标志 0 后，在输入带的第一格印出一个数字 1 并停机，就说程序 P 接受输入字符串 $a_1a_2\cdots a_n$，或者说 $a_1a_2\cdots a_n$ 是可由程序 P 识别的。

P 接受的(或 P 识别的)语言 $L(P)$ 是 P 可接受的所有输入串的集合(这个集合中字符串的个数可能是有穷的，也可能是无穷的)。对于不属于 P 接受的语言 $L(P)$ 中的输入串，P 或者在输出带上印出一个异于数字 1 的符号并终止，或者永远不停机。

下面是两个 RAM 程序的例子。其中第一个例子计算一个函数，第二个例子识别一个语言。

(1)考虑如下定义的函数 $f(n)$：

$$f(n)=\begin{cases} n^n & n>0 \\ 0 & n\leqslant 0 \end{cases}$$

计算 $f(n)$ 的算法描述如下。

```
//计算 f(n)的算法//
procedure FUNCTION(r1)
  begin
    if r1≤0 then write(0);
    else
      begin
        r2 ← r1;
        r3 ← r1 - 1;
        while r3 > 0 do
          begin
            r2 ← r2 * r1;
            r3 ← r3 - 1;
          end;
        write(r2);
      end
  end.
```

与之相对应的 RAM 程序如表 1.5 所示。其中变量 r_1，r_2 和 r_3 分别存放在 RAM 的内存储器 R_1，R_2 和 R_3 中。

表 1.5　与(1)对应的 RAM 程序

标　号	RAM 指令		算法中的相应语句
	READ	1	read (r_1)
	LOAD	1	if $r_1 <= 0$ then write (0)
	JGTZ	pos	
	WRITE	=0	
	JUMP	endif	
pos:	LOAD	1	$r_2 \leftarrow r_1$
	STORE	2	
	LOAD	1	$r_3 \leftarrow r_1 - 1$
	SUB	=1	
	STORE	3	
while:	LOAD	3	while $r_3 > 0$ do
	JGTZ	continue	
	JUMP	endwhile	
continue:	LOAD	2	$r_2 \leftarrow r_2 * r_1$
	MULT	1	
	STORE	2	
	LOAD	3	$r_3 \leftarrow r_3 - 1$
	SUB	=1	
	STORE	3	
	JUMP	while	
endwhile:	WRITE	2	write (r_2)
endif:	HALT		

(2)考虑一个 RAM 程序，它接受字母表 $\Sigma = \{1, 2\}$ 上的一个语言 L_{12}，该语言包含所有由同样多个 1 和 2 组成的符号串。

程序的核心思想是，依次从输入带上读入符号 x 到寄存器 R_1 中，并且在寄存器 R_2 中记录到目前为止所输入的数字 1 和 2 的个数之差 d。当输入带头读到结束标志符 0 时，程序检查 d 是否为 0。如果 $d=0$，则在输出带上打印数字 1 并停机；否则，机器不输出任何结果。

接受该语言的算法可以描述如下：