第 1 章　算法及其可视化教学支持系统

江畔何人初见月？江月何年初照人？
[唐]张若虚,《春江花月夜》

算法权威著作——《算法导论》[1]的前言以阐明算法具有悠久历史作为开篇：在计算机诞生之前早就有了算法，而在计算机诞生之后，人们创造了更多的算法，算法是计算的核心。这句话深刻地表达了算法作为一种求解问题的科学方法，其历史远比计算机悠久，也就意味着算法思维、思想和方法有着深广的科学内涵，需要以超越程序设计、数据结构乃至计算机的视野来认识、学习和探索。本章首先介绍欧几里得（Euclid）在公元前 300 年提出的，至今仍具有重要应用的最大公约数（GCD）算法，以使读者在一开始就能以一个颇具代表性的具体算法切实地建立关于算法的初步认识。该算法既简洁易理解，又蕴含智慧不失深奥，是开启算法学习之旅的良好实例。然后介绍算法的专业化定义与描述。最后介绍基于计算教学论[2]的可视化算法教学模式及相应的算法可视化工具和支持系统，该教学模式在广度、深度和效率上都远优于传统的算法教学模式。

1.1　初识算法：Euclid GCD 算法

本节将介绍号称人类历史上第一个算法的 Euclid GCD 算法。为此，首先循着除法定理、整除、约数、GCD、算术基本定理等较为严谨的数学路线，引出直观地求解 GCD 的因子分解方法，并说明该方法的计算不可行性，然后给出 Euclid GCD 定理，以及由此带来的高效率的、2300 年后的今天还在焕发光芒的 Euclid GCD 算法。

1.1.1　GCD 及因子分解方法

在初等数学中，我们学习过除法定理（证明从略）。

定理 1-1（除法定理）　给定任何整数 a 和任何正整数 d，存在唯一整数 q 和满足 $0 \leqslant r < d$ 的唯一整数 r，使 $a = qd + r$。

其中，a 为被除数，d 为除数，$q = \lfloor a/d \rfloor$ 称为除法的商，r 称为除法的余数，r 又称为 a 对 d 的模运算的结果，记为 $r = a \bmod d$。

符号 $\lfloor \cdot \rfloor$ **为底函数符号**，符号 $\lceil \cdot \rceil$ **为顶函数符号**。底函数和顶函数是算法领域中常用的两个特别的函数，因而我们在此给出其定义和示例。

定义 1-1（底函数）　不大于 x 的最大整数称为 x 的底函数，记为 $f(x) = \lfloor x \rfloor$。

例如，$\lfloor 2.0 \rfloor = 2$，$\lfloor 2.1 \rfloor = 2$，$\lfloor 2.9 \rfloor = 2$，$\lfloor 3.0 \rfloor = 3$。在一般编程系统中常以 floor 实现底函数。

定义 1-2（顶函数）　不小于 x 的最小整数称为 x 的顶函数，记为 $f(x) = \lceil x \rceil$。

例如，$\lceil 2.0 \rceil = 2$，$\lceil 2.1 \rceil = 3$，$\lceil 2.9 \rceil = 3$，$\lceil 3.0 \rceil = 3$。在一般编程系统中常以 ceil 或 ceiling

实现顶函数。

定义 1-3（整除与不能整除） 如果除法运算 a/d 的余数 $r=0$，则称 d 整除 a，记为 $d \mid a$；如果 $r \neq 0$，则称 d 不能整除 a，记为 $d \nmid a$。

定义 1-4（约数） 如果 $d \mid a$，则称 d 是 a 的约数。

例如，1、2、3、4、6、12 都是 12 的约数。不失一般性，我们设 $a \geqslant 0$，即 $a \in \mathbb{N}=\{0,1,2,\cdots\}$ 为自然数，有关结论也适用于 $a<0$ 的情况。显然，对于任何的 $a>0$，a 有 1 和 a 本身这两个约数，它们称为 a 的平凡约数。由于 $0=0\cdot d+0$，因此任何正整数 d 都是 0 的约数。

定义 1-5（素数与合数） 若 $a>1$ 且只有平凡约数，则称 a 为素数（Prime），也称为质数；若 $a>1$ 且有不平凡的约数，则称 a 为合数（Composite）。

1、0 和负数都不是素数也都不是合数。

定义 1-6（公约数） 如果 $d \mid a$ 且 $d \mid b$，则称 d 为 a 和 b 的公约数。

定义 1-7（最大公约数） 称 a 和 b 的公约数中的最大者为 a 和 b 的最大公约数（Greatest Common Divisor，GCD）。

例如，30 有约数 1、2、3、5、6、10、15、30，因而 30 与 12 的 GCD 为 6。

定义 1-8（最大公约数问题） 称求解 GCD 的问题为最大公约数问题。

因为任何正整数 d 都是 0 的约数，所以若 $a>0$，$b=0$，则 a 和 b 的 GCD 就是 a。

算术基本定理，也称为唯一素因子分解定理（证明从略），给了我们一个求解 GCD 的直觉方法。

定理 1-2（算术基本定理，即唯一素因子分解定理） 任何一个大于 1 的自然数 $\mathbb{N}$，或者自身为一个素数，或者可表示为一系列素数（素因子）的积，且这种表示在不考虑素因子次序的情况下是唯一的，即 $\mathbb{N}=p_1^{m_1}p_2^{m_2}\cdots p_n^{m_n}$，其中 $p_1,p_2,\cdots,p_n$ 为素因子，$m_1,m_2,\cdots,m_n$ 为大于或等于 1 的整数。

例如，$12=2^2\times 3^1$，$30=2^1\times 3^1\times 5^1$。

根据定理 1-2，我们可以直觉地构造一种如下所示的求解 GCD 的因子分解方法。

注：我们将它称作“方法”而非“算法”，因为它并不完全具备算法的特性（1.2.2 节将详述算法的特性）。

求解 GCD 的因子分解方法如下。

第 1 步 对给定的正整数 a 和 b 进行唯一因子分解，并将其中的各因子按照从小到大的顺序排列：$a=p_{a1}^{m_{a1}}p_{a2}^{m_{a2}}\cdots p_{an}^{m_{an}}$，$b=p_{b1}^{m_{b1}}p_{b2}^{m_{b2}}\cdots p_{bn}^{m_{bn}}$。

第 2 步 依次对 a 和 b 的因子进行比较，找出公共因子及次数：$q_1^{m_1},q_2^{m_2},\cdots,q_k^{m_k}$。

第 3 步 将各公共因子按次数求积，便可得到 GCD：$\mathrm{GCD}(a,b)=q_1^{m_1}q_2^{m_2}\cdots q_k^{m_k}$。

例如，$98=2\times 7^2$，$56=2^3\times 7$，因而 $\mathrm{GCD}(98,56)=2\times 7=14$。

又如，$168=2^3\times 3\times 7$，$180=2^2\times 3^2\times 5$，因而 $\mathrm{GCD}(168,180)=2^2\times 3=12$。

尽管上述求解 GCD 的因子分解方法中第 1 步的因子分解和第 2 步的因子比较还需要进一步明确为更具体的操作，但我们确信这种具体的操作是存在的，因而方法的正确性没有问题。但问题是第 1 步中整数的因子分解目前尚未找到高效率的算法，如对 1024 个二进制位的整数进行因子分解以目前的计算机性能需要以世纪来衡量的时间，因而实际计算是不可行的。而 Euclid 提出的求解 GCD 的算法是一个高效率，因而是实际计算可行的算法。

1.1.2　Euclid GCD 算法

早在公元前 300 年，著名的古希腊数学家、几何学之父 Euclid 就在其被认为是人类历史上最成功的教科书《几何原本》中，提出了一个高效率求解 GCD 的算法。这也为“在计算机诞生之前早就有了算法”的观点提供了确凿的证据。该算法被称为 Euclid GCD 算法，它是人类历史上最早的算法之一。令人惊叹的是，该算法至今仍然具有很重要的应用，它是 1977 年提出、至今应用广泛的 RSA 算法的重要组成部分，RSA 算法是奠定今天网络安全的最优秀的密码学算法之一。Euclid GCD 算法有时也被形象地称为辗转相除法。

Euclid GCD 算法来自如下的 Euclid GCD 定理（证明从略）。

定理 1-3（Euclid GCD 定理）　对于给定的自然数 $a>0$ 和 $b\geqslant 0$，有：

（1）当 $b=0$ 时，$\mathrm{GCD}(a,0)=a$。（2）当 $b>0$ 时，$\mathrm{GCD}(a,b)=\mathrm{GCD}(b,a \bmod b)$。

有了定理 1-3，我们就可以构造求解 GCD 的 Euclid 算法。我们先将算法理解为“以步骤表示的求解问题的方法”，1.2 节再详细讲述算法的概念和定义。

算法 1-1 用了较为专业的伪代码进行描述，而且顺序标注了行号，以方便解释。伪代码借用了一些 C 语言中的元素，如 while 循环、赋值语句等，但也有更自然的描述，如 b ≠ 0、以 end while 而不是花括号结束循环体等。1.3.3 节将详细介绍算法的伪代码描述方法。

算法 1-1　Euclid GCD 算法

输入：整数 a > 0,b ⩾ 0

输出：a 与 b 的 GCD

1．while　b ≠ 0

2．　　r = a mod b；　a = b；　b = r

3．end while

4．return a

为了直觉地理解上述算法，我们可以设计一个表格，对给定的数据按照算法的步骤进行推演，如表 1-1 所示。表 1-1 中列出 a 除以 b 的商 $q=\lfloor a/b \rfloor$ 是为了帮助理解，其实 Euclid GCD 算法是不需要计算 a 除以 b 的商的。

表 1-1　Euclid GCD 算法的计算示例

No	a	b	$q=\lfloor a/b \rfloor$	r
1	98	56	1	42
2	56	42	1	14
3	42	14	3	0
4	14	0		

注：用给定输入数据以“手算”表格推演算法的方法，是学习和理解算法很实用且有效的方法。

1.2　算法的定义

本教材致力于讲述算法，因此我们尽早地在本节给出算法的定义。本节将首先阐述算法是一种求解问题的科学方法，是在理论方法和实验方法之外的第三种科学方法；其次给出通俗意义上的算法的克努特定义；最后给出严谨意义上的算法的图灵机定义。

1.2.1 算法是一种求解问题的科学方法

1.1 节中的 Euclid GCD 算法示例了“以步骤表示的求解问题的方法”这个算法特征，尽管该算法不能一下子给出 a 和 b 的 GCD，但是它能够通过一轮计算将求解 a 和 b 的 GCD 问题转化为求解 b 和 $a \bmod b$ 的 GCD 问题，而求解 b 和 $a \bmod b$ 的 GCD 问题是一个比求解 a 和 b 的 GCD 问题规模要小且多数情况下小很多的问题。这里面就蕴含着算法求解问题的最重要思想之一，即问题分解思维。当问题规模小到一定程度，对于 GCD 来说就是小到 $b=0$ 的程度时，即可停止分解过程直接获得解。

算法方法是一种问题的程序化解决方案[3]，这种解决方案本身并不直接是问题的答案，也不直接给出问题的答案，而是获得答案的一些指令及其执行流程。这些指令和流程并不是随意的，而是由严谨的**图灵机**（参见 1.2.3 节）来定义的。这使以算法为核心的计算机科学有别于传统的理论数学，后者主要考虑对某个问题是否有解进行证明，如果有解，则再对解的性质进行研究。

算法已被广泛认可为理论方法和实验方法之外的第三种科学方法。牛顿于 1687 年发现的如式（1-1）所示的万有引力定律，是运用理论方法探索科学的一个典型例子。该理论准确地解释了行星在围绕太阳的椭圆轨道上运行且太阳处在椭圆的一个焦点上等一系列天体运行规律，并成为今天火箭发射、航天探索的科学实践依据。

$$F = G\frac{Mm}{r^2} \tag{1-1}$$

爱因斯坦于 1915 年提出的如式（1-2）所示的广义相对论，是运用理论方法探索科学的另一个典型例子。该理论准确地解释了水星近日点的进动现象，成功地预测了光线在引力场中的弯曲、引力红移及引力波等宇宙现象。

$$\boldsymbol{R}_{\mu\nu} - \frac{1}{2}R\boldsymbol{g}_{\mu\nu} + \boldsymbol{g}_{\mu\nu}\boldsymbol{\Lambda} = \frac{8\pi G}{c^4}\boldsymbol{T}_{\mu\nu} \tag{1-2}$$

2015 年 9 月 14 日引力波的探测发现，是运用实验方法探索科学的一个典型例子。这个探测是由 LIGO 和 Virgo 研究团队，利用在美国路易斯安那州的列文斯顿（Livingston）和华盛顿州的汉福德（Hanford）建造的两个臂长达 4km 的引力波探测器（见图 1-1）联合完成的。

图 1-1 列文斯顿（Livingston）和汉福德（Hanford）的两个引力波探测器[4]

既然算法被赋予了与具有超级智慧的杰出人类才能发现的科学理论方法，以及借助超级精密和规模的仪器装置才能开展的实验研究方法等同的地位，那么它一定有其深刻的内涵，这也正是本教材致力于揭示的内容，这些算法内容远不只是数据结构课程知识的升级版。

人类之所以如此推崇算法，是因为当为一个问题设计了算法后，就可以在不再需要人的智力的情况下机械和自动地演算求解过程，并获得问题最终的求解结果。人类也的确发明了

被称为计算机的、可以自动执行算法的机器，它能够以比人工快得多和精确得多的方式执行算法，尤其对于重复性操作不知疲倦、不厌其烦、不出错误。

然而，用“以步骤表示的求解问题的方法”描述算法仅是一种大众化的、非学术性的描述，我们需要对算法进行严谨的定义，以使其成为一种有理性内涵的、揭示问题及其求解的本质特征的科学方法。

1.2.2　算法的克努特定义

公认的通俗意义上的算法定义，是唐纳德·克努特（D. E. Knuth）在其被誉为计算机领域的圣经的著作《计算机程序设计艺术》第 1 卷中的定义[5]。

定义 1-9（算法的克努特定义）　算法是一组有穷规则的集合，这些规则给出了求解特定类型问题的运算序列。它具有 5 个重要特性，即有穷性（Finiteness）、确定性（Definiteness）、有效性（Effectiveness）、输入（Input）和输出（Output）。

下面通过说明这 5 个重要特性对上述定义进行细致解释。

1．有穷性

一个算法必须保证在执行有限个步骤之后结束，不能终止的计算过程不能算作算法。

例如，求圆周率 π 的一种古老方法是用正多边形的边长逼近圆的边长（见图 1-2），这个过程如果不人为地设定一个截止条件，就是一个无穷的过程，而不是一个算法。

注：如果一个计算过程不是有穷的，但满足其他 4 个算法特性，则称其为一个计算方法[5]。

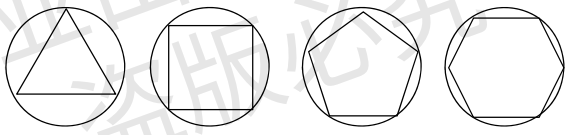


图 1-2　正多边形逼近法求圆周率 π

2．确定性

算法的每个步骤都必须有确切的、无二义性的定义。

“求解 GCD 的因子分解方法”尽管不是一个算法，但是其第 1 步却是一个确定性的步骤。如果不限定 a 和 b 为正整数，则当 a 或 b 为 0 或负数时，“唯一因子分解”就没有定义。如果不限定“各因子按照从小到大的顺序排列”，则因子分解的表达式就不是唯一的。这都会使该步骤具有不确定性。算法 1-1 中的“输入：整数 $a>0, b\geqslant 0$”也是确定性的描述，尽管“$a>0$”仅是为了好理解定义的，并不是必须定义的，但是“$b\geqslant 0$”是必须定义的，因为如果 $b<0$，$a \bmod b$ 就没有定义，会造成不确定性。

3．有效性

算法中的所有运算或操作必须充分基本，即原则上可以由人用笔和纸在有限的时间里准确地完成。

“求解 GCD 的因子分解方法”的第 1 步和第 2 步尽管具有确定性，但是不具有有效性。因为“对给定的正整数 a 和 b 进行唯一因子分解”和“依次对 a 和 b 的因子进行比较，找出公共因子及次数”都不是基本操作，需要进一步细化。

注：有效性不能教条地理解。例如，RSA 算法中要调用扩展 Euclid GCD 算法，而扩展 Euclid GCD 算法是一个很明确的算法，因而在 RSA 算法的主算法描述中，可以将该算法看作一个基本操作。

4．输入

一个算法可以有 0 个或多个输入，这些输入可以是算法开始之前已经被赋值的状态参数，也可以是运行过程中动态赋值的量。但在绝大多数情况下是算法开始运行时输入的数据。0 输入表示算法自身已确定了初始运行条件。

5．输出

算法必须有 1 个或多个输出，以反应对输入或预置数据计算或处理的结果。没有输出的算法是毫无意义的。

有了克努特关于算法的定义后，我们可以用图 1-3 说明用算法求解问题的基本模式：首先针对给定的问题设计求解算法；然后在计算机里实现算法；最后给在计算机里实现的算法提供具体的输入数据，计算机针对输入数据运行算法，并输出结果。

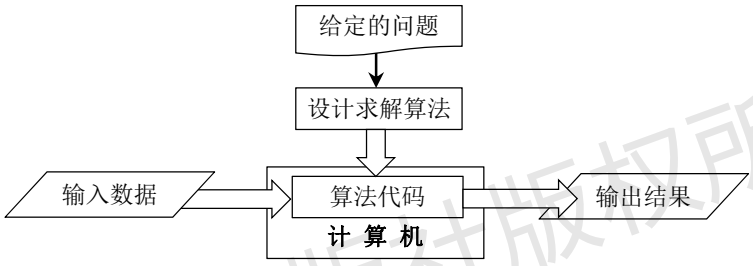


图 1-3　用算法求解问题的基本模式

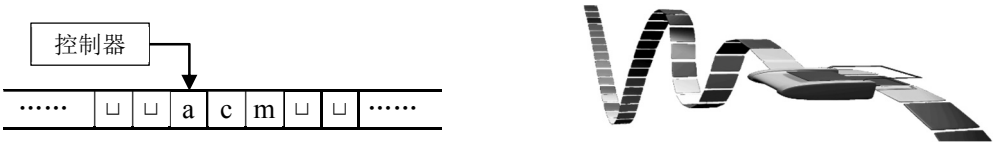
算法的克努特定义是一个通俗易理解的定义，但是正如克努特自己所说[5]，这个定义并不是一个严谨的数学化定义，因而不能对算法这个概念给出严格的揭示和限定。严谨意义上的算法定义是基于图灵机的定义。

1.2.3　算法的图灵机定义

要严谨地定义算法，首先要定义计算模型。图灵机模型是所有计算模型中最好理解，因而被广泛接受的模型。1936 年 12 月 12 日，艾伦·图灵（Alan M. Turing）在伦敦数学学会作了题为“论可计算数及其在判定性问题中的应用”（*On Computable Numbers, with an Application to the Entscheidungsproblem*）的报告。在该报告中，他提出了一个被称为“通用计算引擎”（Universal Computing Engine）的抽象数学装置，也就是后来人们以他的名字命名的图灵机，12 月 12 日也就成为图灵机的诞生纪念日[6]。为了纪念图灵的这一伟大贡献，人们将他誉为“计算机科学之父”，美国计算机学会（ACM）于 1966 年以他的名字设立了号称计算机界诺贝尔奖的“图灵奖”，以表彰在计算机科学领域中做出奠基性贡献的计算机科学家。

本节将扼要地介绍图灵机及其数学定义，以引出关于算法的严谨定义，但不展开解释。想要深入学习图灵机可阅读专门的计算理论教程，如《计算理论导引》[7]。

图灵机是一个抽象的计算模型，图 1-4（a）所示为图灵机的原理示意图，图 1-4（b）所示为图灵机的艺术化示意图。图灵机包括一个控制器、一个读写头和一个无限长的带子。控制器包含一系列的状态。带子由一系列格子组成，每个格子中可以包含一个字母或空白字符。从图 1-4 中可以看到，强大的计算机的理论模型竟然如此简单，实在令人惊奇。



（a）图灵机的原理示意图　　（b）图灵机的艺术化示意图[8]

图 1-4　图灵机的示意图

图灵机有如下严谨的形式化定义。

定义 1-10（图灵机）[9]　图灵机是一个 7 元组$(Q,\Sigma,\Gamma,\sqcup,q_0,F,\delta)$，其中：

Q 是有穷的状态集。

Σ 是有穷的输入字母表。这里的“字母”是广义的，如常用的$\Sigma=\{0,1\}$就是以 0 和 1 作为“字母”的。

Γ 是有限的带子字母表，$\Gamma\supset\Sigma$。Γ 中除包含不属于 Σ 的空白字符⊔以外，还可能包含其他不在 Σ 中的字母，这些字母是在计算过程中生成的。

⊔是空白字符，$\sqcup\in\Gamma$，但$\sqcup\notin\Sigma$。

$q_0\in Q$ 是起始状态。

$F\subseteq Q$ 是结束状态集，也称为停机状态集。

$\delta:Q\times\Gamma\to Q\times\Gamma\times\{\mathrm{L},\mathrm{R},\mathrm{N}\}$ 是转移函数。

尽管计算机的理论模型像图 1-4 那样简单，但是其形式化定义却是如此的深奥，因而计算机如此强大也就不足为奇了。

图灵机的核心是转移函数$\delta:Q\times\Gamma\to Q\times\Gamma\times\{\mathrm{L},\mathrm{R},\mathrm{N}\}$，其中 L、R 和 N 分别表示读写头左移、右移和原地不动。例如，$\delta(q,\mathrm{a})=(r,\mathrm{b},\mathrm{L})$表示处于 q 状态的图灵机读到字母 a 后进入状态 r，读写头写下字母 b 以取代 a 并左移一格。

图灵机的执行过程，也就是计算方式如下。开始时带子上有 n 个连续的输入字母序列$a_1,a_2,\cdots,a_n$，其余格子中为空白字符，图灵机处于状态q_0，读写头处在含有第 1 个字母a_1的格子中。当计算开始时读写头读取当前格子中的字母a_1，图灵机计算转移函数$\delta(q_0,a_1)$，根据其返回结果改写控制器的状态和第 1 个格子中的内容并移动读写头。此后，图灵机会一直按照转移函数决定的操作一步一步地执行下去，当到达某个结束状态$q_f\in F$时，计算过程结束。此时带子上的字母序列便是计算的输出结果。

注：由于 Γ 中包含空白字符⊔，当处于状态q_x的图灵机读到空白字符⊔时，转移函数$\delta(q_x,\sqcup)$依然有定义，因此仍然可以计算出接下来的操作。如果处于状态q_x的图灵机读到字母 y 时，$\delta(q_x,\mathrm{y})$没有定义，则该次计算失败。如果图灵机的计算永远到达不了结束状态，则这个计算也是失败的。

这个看起来简单普通的计算过程，却是现代计算机的理论基础，蕴含着计算机的强大计算能力，实在出人意料。

图灵机的严谨定义就是算法的严谨定义。

定义 1-11（算法的图灵机定义）　可执行到结束状态的一个图灵机就称作一个算法。

按照这个定义，每设计一个算法就需要设计一个图灵机。对应到现实世界，也就意味着要为每个算法设计一台计算机。这种模式的实际应用显然是不可行的。图灵进一步从理论上解决了这个问题，他证明可以设计一种通用图灵机，它能够模拟执行任何其他图灵机上的计算。这一结论为现代通用计算机奠定了理论基础。

1.3 算法的描述方法

算法既然有严谨的图灵机定义，也就必须有严谨的描述方法。从理论上说，要描述一个算法就应该描述其对应的图灵机，即给出对应的图灵机（定义 1-10）的 7 元组。显然这个意义上的算法描述过于底层和烦琐，我们应该有更高层次的、既便于机器理解又便于人理解的描述方法。这些更高层次的描述也不能随意，应该以遵循算法的克努特定义（定义 1-9）为基本原则。本节将介绍算法的常用描述方法，即自然语言描述方法、流程图描述方法、伪代码描述方法，以及现代版 C++描述方法，并分别说明其适用的范围。

1.3.1 算法的自然语言描述方法

用自然语言描述算法是一种很容易为人所理解的方法，但还是要表达出克努特所列的算法的 5 个特性。下面给出了 Euclid GCD 算法的自然语言描述。

算法 1-2　Euclid GCD 算法的自然语言描述

1．输入大于或等于 0 但两者不能同时为 0 的整数 a 和 b。
2．判断 b 是否为 0，如果不为 0，则转第 3 步；否则，转第 4 步。
3．a 对 b 取余数，将结果赋值给 r，b 赋值给 a，r 赋值给 b，转第 2 步。
4．输出 a，算法结束。

算法 1-2 以序号对算法过程进行步骤化；第 1 步和第 4 步明确给出输入和输出；以严谨的语言描述来表达操作的确定性，如第 1 步的“输入大于或等于 0 但两者不能同时为 0 的整数 a 和 b”；以“a 对 b 取余数，将结果赋值给 r，b 赋值给 a，r 赋值为 b”表达操作的有效性；以“转第 n 步”的方式表达流程。

设计算法的最终目的是用计算机运行算法求解问题，这需要尽量将算法表达为机器可读的形式。显然用自然语言描述的算法到机器可读的距离较大，因此在实践中常用自然语言描述算法的设计思想，而具体的算法描述则多用后面将要介绍的伪代码或程序设计语言来实现。Euclid GCD 算法的基本思想就是 Euclid GCD 定理，因此可以用如下的自然语言描述其基本思想。

算法 1-3　Euclid GCD 算法的思想描述

根据 Euclid GCD 定理，即 $\mathrm{GCD}(a,0)=a$，$\mathrm{GCD}(a,b)=\mathrm{GCD}(b,a \bmod b)$，可以设计一个高效率求解 GCD 的算法，即 Euclid GCD 算法。

注：算法的思想描述主要描述算法最关键的理论和技术，因此不需要太过严谨。

1.3.2 算法的流程图描述方法

流程图是一种直观的算法描述方法，它使用图形化的符号描述算法中的基本操作，使用带箭头的线段表达算法的流程。

图 1-5 所示为 Euclid GCD 算法的流程图，表 1-2 所示为常用的基本流程图符号及其意义。显然，流程图最大的特点是可以非常直观地表达算法的流程，即各基本操作之间的先后关系。但流程图也有明显的短板，即修改很耗时、费力。当算法或其某个部分具有复杂的逻辑时，可使用流程图描述，以帮助直观地设计和理解流程。

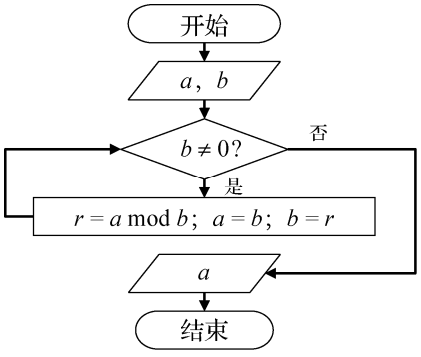
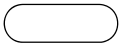
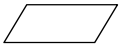
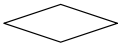
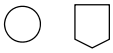


图 1-5　Euclid GCD 算法的流程图

表 1-2　常用的基本流程图符号及其意义[10]

符　号	意　义
	圆角矩形框表示流程的开始和结束
	平行四边形框表示输入、输出操作
	矩形框表示基本运算或操作
	菱形框表示根据条件判断执行不同的分支
	圆形和锥形符号用于页内和页间的连接

1.3.3　算法的伪代码描述方法

描述算法最权威且被普遍采用的方法是伪代码描述方法。

定义 1-12（伪代码）　伪代码是介于自然语言与程序设计语言之间的一种结合文字和符号的算法描述方法。它注重算法本质性思想和技术的描述，忽略程序设计语言中一些烦琐的语法规则与细节，因而是一种既具有严谨性又便于书写和理解的算法描述方法。

算法的伪代码描述方法是一种注重表达算法思想的方法，并不存在一种公认的伪代码标准。为了帮助读者掌握伪代码描述的基本要领，我们将算法的伪代码描述规范总结如下。

1．说明简化的算法中文或英文名称。
2．说明算法的输入和输出。
3．顺序标记行号。
4．赋值语句：使用=或←进行变量赋值。
5．变量可以带有下标，如 x_1、y_i。
6．可以使用方括号表示数组和元素的下标，如 $x[1]$、$y[i]$。
7．必要时可以使用数学符号，如 α、$\varnothing$。
8．必要时可用由数学公式表示的运算，如 $x=\frac{b}{a^2}$。
9．使用易理解的英文单词描述操作，如交换 a 与 b 的值可描述为 swap(a, b)。
10．使用直观易理解的符号描述操作，如交换 a 与 b 的值可描述为 $a \leftrightarrow b$。
11．使用直观易理解的数学函数，如 $\sin\theta$。
12．使用//表示注释，如 swap(a, b) //交换 a 和 b 的值。
13．使用 return 结束算法，必要时返回结果。
14．使用 if 语句表示分支，可以带有 then、else 和 else if。
15．for 循环：for i = 1 to (down to) n step k //step 为 1 时可省略。
16．循环还可有 while 循环、do … while 循环、repeat … until 循环等。
17．使用缩格表示操作的不同层次。
18．复合语句：
　a）当分支或循环体有一个语句时，可以不用复合语句标识。
　b）可以使用{ … }作为复合语句界定符。
　c）可以使用 begin … end 作为复合语句界定符。
　d）可以使用 end if、end for 和 end while 作为复合语句界定符。

根据上述规范，我们可以将 Euclid GCD 算法重新表达为如算法 1-4 所示的规范化伪代码。

算法 1-4　Euclid GCD 算法的规范化伪代码

输入：整数 $a>0, b\geq 0$　// a 和 b 不能同时为 0
输出：a 与 b 的 GCD
1．while　$b\neq 0$
2．　　r = a mod b ; a=b; b=r
3．end while
4．output a

1.3.4 算法的现代版 C++描述方法

学习和设计算法的重要目的是将问题的求解交给计算机自动化运算和处理，因此从学习的角度来看，只有具体实现了的算法才算是彻底掌握了的算法；从实际应用的角度来看，只有具体实现了的算法才算是真正解决了实际问题、实现了价值的算法。因此，算法实现在学习和设计算法过程中是体现最终成果的一环，是必须完成的任务。

然而算法实现却是算法教与学中的一个巨大难题。一方面，大部分传统算法教科书及课程主要用伪代码描述算法，而对于很多学生来说，伪代码和算法实现之间存在着难以跨越的鸿沟；另一方面，有些教科书及课程采用了 C/C++或 Java 程序设计语言描述算法，这些语言过于基础和烦琐，表达能力过弱，将算法中的核心思想淹没在了过于冗赘的代码中，这给教学过程中的书面和口头表达带来了巨大的负担，在实际的教学运用中是不可行的。

幸运的是，现代版 C++（Modern C++）的诞生恰当地解决了这个问题。现代版 C++指的是 C++1x 标准的 C++及对应的标准函数库，包括 C++11 及后续的补充加强标准 C++14、C++17[11]等。相较传统的 C++98/C++03 标准，C++11 是一个里程碑式的标准。鉴于 C++11 标准的实质性重大改进，它被 C++创立人 Bjarne Stroustrup 描述为“感觉像一个新的语言”[12]。运用现代版 C++的编程功能，如自动数据类型推断 auto、基于范围的 for 循环（range-based for loop）、lambda 表达式支持的匿名函数、列表初始化、伪随机数发生器等，结合 C++的泛型编程功能，如以模板方式实现的 vector、pair、queue、set 等泛型类和 swap、sort 等泛型函数，以及迭代器等功能，能够以简洁性与伪代码相媲美的专业 C++代码描述和实现算法。即使像 TSP 问题的 DP 算法这样以子集索引子问题的算法，也能运用上述现代版 C++编程功能以很简洁的方式描述和实现。这填平了算法思想、伪代码和算法实现之间的鸿沟，使算法实现这个算法教与学中的巨大难题得到了恰当的解决。

算法 1-5 给出了 Euclid GCD 算法的 C++代码。尽管该代码并未展现出现代版 C++的新特征，但是我们也能看出它比较简洁，不影响人们对算法的理解。今后许多算法的 C++描述和实现会用到现代版 C++的重要特征，届时我们会给出相应的介绍。

算法 1-5　Euclid GCD 算法的 C++代码

```
1．int EuclidGCD(int a, int b)
2．{
3．        int r;
4．        while (b) {
5．                r = a % b; a = b; b = r;
6．        }
7．        return a;
8．}
```

注：本教材着力以现代版 C++描述和实现算法，主要目的是帮助读者高效率地开展算法学习，快速地进行算法实验。现代版 C++对描述规模较小的算法具有优越性，因此特别适用于描述教学性的算法。但是，读者需要知道，描述算法最权威且被普遍采用的方法仍然是伪代码描述方法，且 C++的强大在于它是编写操作系统级大型软件系统的程序设计语言。我们也将在必要的时候给出一些算法的伪代码描述。

1.3.5 设计算法求解问题的基本过程

设计算法求解问题是计算机科学与技术专业领域中的基本任务之一，描述算法仅是其中

的一项子任务。设计算法求解问题的基本过程是一个系统化的过程，该过程可分解为如下的 6 个子过程。

1．问题定义

对要求解的问题进行精确和严谨的定义是设计算法求解问题的良好开端。克努特定义的算法的 5 个特性要求待求解的问题也必须是与这 5 个特性相容的问题。因此，理想的问题定义是形式化水平上的问题定义，即以数学级严谨的语言和严密的逻辑给出问题的表述，这通常需要进行一些符号化定义和公式化描述，还需要对问题输入和问题所涉范围进行严谨约束。问题定义不是一蹴而就的，很多时候需要经过多次推敲，才能从开始的完全非形式化自然语言描述逐步演化到最终严谨的形式化描述。

2．算法设计思想描述

算法设计思想是算法设计的核心，这里的思想指的是对具体问题设计的算法中采用的关键方法和技术。算法设计是一种人类的智力活动，有些算法设计思想的提出确实需要人类的智慧，如 Euclid GCD 算法、快速排序算法、RSA 算法等。但是大部分算法设计是有章可循、有法可依的。即使前述体现人类智慧的算法设计，其某些环节也遵循了一般性的算法设计方法。讲述这些一般性的算法设计方法是本教材的主体内容，这主要包括穷举法、递归法、分治法、动态规划法、贪心法、回溯法与分支限界法等，它们也被称为算法设计策略或范例。本教材的主要目的就是对这些一般性的算法设计方法及其算法设计实例进行介绍，使读者通过学习成熟的算法设计方法，积累前人成功的算法设计经验，提升自身的算法设计水平，激发自身的算法创造能力。

3．算法的伪代码描述

算法设计思想提出后，需要以伪代码方式将其表达为严谨的算法。伪代码描述方法前文已详细介绍过，此处不再赘述。

4．算法的正确性证明

只有正确的算法才是有价值的算法，因此进行算法的正确性证明是表明算法有价值的必要步骤。由于算法以步骤方式求解规模不断增大直到无穷大的问题，而问题的规模又可以用自然数来衡量，因此数学归纳法是算法的正确性证明很自然且很贴切的方法。注意，通过一些输入数据实例在计算机上验证算法程序能否正确执行，不能算作严格意义上的算法正确性证明。算法的正确性证明也是有一定挑战性和难度的过程，本教材一般不讨论算法的正确性证明，直接认可所讲述算法的正确性。感兴趣的读者可参考《算法导论》[1]学习算法的正确性证明思路、技巧和方法。

5．算法复杂度分析

对于一个正确的算法，其运行效率，即运行所需的时间和空间资源就成为其最关键的衡量指标。运行效率的专业术语为复杂度。算法及其设计方法的多样性也带来了算法复杂度分析方法的多样性。我们将在第 3 章对算法复杂度分析的基本概念和方法进行介绍，并在随后各章的算法讲述中，穿插讲述具体算法的复杂度分析方法。

6．算法的实现及运行测试

前文已述及，以程序设计语言实现算法，并使之在具体的计算机上运行，以帮助人类解决实际问题是设计算法的根本目的。因此，设计算法的最后一个步骤就是算法的实现，这离

不开运行测试环节。尽管运行测试不是证明算法正确性的方法，但却是算法程序可信和健壮运行的保证。因此，应该设计充分的测试数据样例，对算法程序进行全面的测试。运行测试还有一个功能，即对算法的运行效率进行实验分析，这通常是为了比较求解同一问题的不同算法的性能，一般需要在不同分布的较大甚至很大的数据集上进行。

1.4 可视化算法学习的支持工具

算法课程中会介绍很多算法设计实例，算法设计实例通常会有较复杂的逻辑，涉及一些数据结构及其操作，还涉及算法实现。传统教科书式和幻灯片式的图形表达不能对变化的输入进行算法过程演示，并且设计费时、烦琐，这些都给算法的教与学带来许多困难，严重地影响教学效果。为解决这一问题，作者提出了计算教学论[2]（Computational Didactics，CD），并发展了算法可视化（Algorithm Visualization，AV）技术，为本教材中的绝大部分算法创新性地设计了 CD-AV 演示。为使这些 CD-AV 演示具有高可用性，作者还开发了相应的支持和承载系统，即计算机辅助的算法教学系统（Computer Assisted Algorithm Instruction System，CAAIS[13]）。以 CAAIS 支持的 CD-AV 演示辅助高效率和更深、更广的算法学习是本教材的特点，也是作者在算法教与学上的创新举措。本节将扼要地介绍 CAAIS 的基本功能和操作。

1.4.1 CAAIS 的基本界面及其功能

CAAIS 基于 HTML5、CSS3 和 JavaScript6 等 Web 技术设计，因此需要在对这些技术支持最充分的谷歌 Chrome 或微软 Edge 浏览器上运行。目前 CAAIS 支持 Windows、Android 和 macOS 上的 Chrome 和 Edge 浏览器，暂不支持 iOS 系统。

图 1-6 所示为 CAAIS 的基本界面，左侧面板底部有“功能”和“演示”两个选项卡。

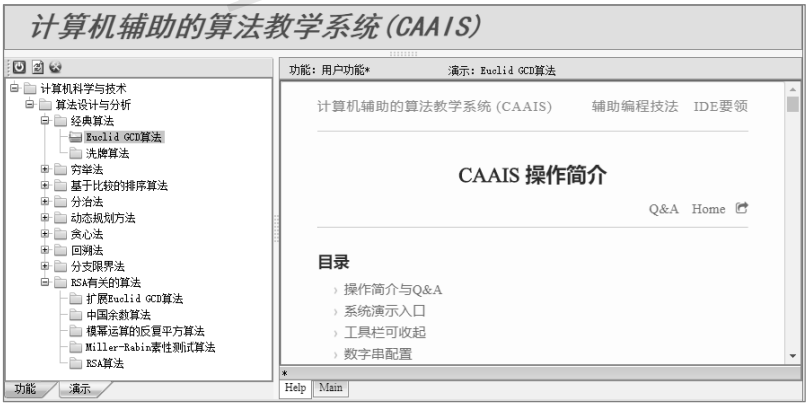


图 1-6 彩图

图 1-6 CAAIS 的基本界面

学生登录系统后，默认显示“演示”选项卡，该选项卡以树形目录列出各算法，双击一个算法即可打开对应的 CD-AV 演示界面。基本界面的右侧面板中是 CAAIS 操作简介，初次使用 CAAIS 的读者应该浏览一下该内容，以掌握系统的基本操作方法。单击“CAAIS 操作简介”标题下面右侧的图标，可以将操作简介显示在单独的浏览器页面中，从而更方便浏览和学习。右侧面板底部有 Help 和 Main 两个选项卡，登录后默认显示的 Help 选项卡中显示 CAAIS 操作简介，单击 Main 选项卡可显示选定功能的界面。

对于普通学生用户，“功能”选项卡中有如图 1-7 所示的功能。单击一项功能，右侧区域将自动切换到 Main 选项卡，并打开该功能对应的界面。图 1-7 中为“查看分数”功能对应的界面，该界面中显示的是用户在进行算法交互练习时所获得的分数。界面顶部的“刷新”按钮用于显示最新的交互操作结果。

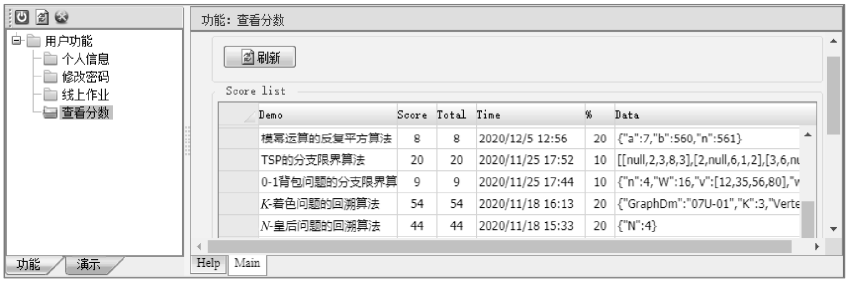


图 1-7
彩图

图 1-7　CAAIS 的“功能”选项卡

1.4.2　算法 CD-AV 演示的基本操作

双击图 1-6 中“演示”选项卡中的一个算法，即可打开该对应的 CD-AV 演示窗口。图 1-8 所示为 Euclid GCD 算法的 CD-AV 演示窗口。该窗口中包括如下的布局和控制设计要素。

图 1-8
彩图

图 1-8　Euclid GCD 算法的 CD-AV 演示窗口

1．窗口布局

窗口由上方的控制面板、左下方的 CD-AV 演示区和右下方的辅助区三个部分组成。

2．控制面板

控制面板中包括典型预置数据、用户数据输入、演示历史顺序（升序或降序）选择、交互控制及动画设置（Quick、Slow 或 Off）等功能模块。注意，在控制面板中选择、输入了数据或修改了状态后，需要单击“更新”按钮才能将信息传递到 CD-AV 演示区。

3．CD-AV 演示区

CD-AV 演示区顶部为功能按钮面板，其中提供了“单步”（Next step）、“一遍”（Next pass，对某些算法可以一次运行外循环中的一遍或某个大步骤）、“运行所有步骤”（Run to the end）、

“暂停”（Stop）、“重置”（Reset）CD-AV 演示控制按钮，以及“交互得分保存”（Save score）、“CD-AV 幻灯片 pdf 打印”（Print slides）和“代码 pdf 打印”（Print code）辅助功能按钮。注意，CD-AV 演示区中的很多界面元素都提供了鼠标指针悬浮提示（Tooltip）功能，图 1-8 中的 Next step 就是对“单步”按钮的提示说明。功能按钮面板的右侧显示算法在当前输入数据下的总步骤数。

CD-AV 演示区主体区域的顶部是标题行，描述 CD-AV 数据子行中各单元格中的量。标题行下面是当前 CD-AV 行。对于 Euclid GCD 这类算法，CD-AV 行包括一个短语子行和一个数据子行。短语子行提供对算法步骤的简洁文字描述；数据子行提供算法运行过程中变量及数据值的显示、状态变化和交互。CAAIS 中 CD-AV 演示的一个突出的特点是以升序或降序的方式将算法执行的所有步骤（历史）对应的 CD-AV 行记录下来。图 1-8 展示的是默认的降序记录的情况。该功能大大方便了对算法执行步骤的连贯学习。CAAIS 还能够将升序演示的所有步骤以幻灯片的方式输出为 pdf 文档，以方便脱离 CAAIS 进行算法教授和学习。许多算法的 CD-AV 行还包括一个图形子行或图形区，我们将在有关的算法中对其进行说明。

4．交互操作

CAAIS 提供了根据一定比例对随机选定的算法步骤中数据子行的变量和数据值进行交互操作的功能。如图 1-8 所示。在输入框中输入数据后，单击交互行末的“提交”（Hand in）按钮，系统将自动判断输入数据的正确性，并在本行右侧给出正确输入数据的统计，功能按钮面板的右侧显示迄今各交互行正确输入数据的合计。当演示结束时，可以单击功能按钮面板中的“交互得分保存”按钮，将交互结果以得分的形式记录到数据库中。

5．辅助区

辅助区中以选项卡的形式给出了算法的代码跟踪页（Code）、算法描述页（Text）和算法 CD-AV 演示帮助页（Help）。有些算法还在辅助区中提供了输入/输出（I/O）页。

代码跟踪页（Code）（见图 1-8）中以彩虹方式展示包括测试函数在内的完整现代版 C++ 代码，更为重要的是，该页提供了随算法执行步骤的代码跟踪功能。当前算法步骤对应的代码以高亮颜色显示，CAAIS 允许一个 CD-AV 步骤对应多行代码，并且以序号标记代码行的执行次序。CAAIS 将上一个步骤对应的代码行以次高亮的颜色显示。代码跟踪功能是 CAAIS 的又一个重要特征，它将算法的执行步骤与现代版 C++代码关联，将算法的步骤解析落地为具体的代码，为算法逻辑的理解提供有力的支持。

算法描述页（Text）中提供了算法设计思想和方法的图文解释，算法 CD-AV 演示帮助页（Help）中提供了关于 CD-AV 演示的特别设计说明和操作要领提示，如图 1-9 所示。

图 1-9（a）彩图

图 1-9（b）彩图

（a）算法描述页（Text）　　（b）算法 CD-AV 演示帮助页（Help）

图 1-9　算法描述页（Text）与算法 CD-AV 演示帮助页（Help）

1.5　使用现代版 C++进行算法实验

前文已述及，算法实验是进行算法学习和运用必须进行的检验，而要进行算法实验必须选择一种适宜的程序设计语言。1.3.4 节中已经论述过，本教材主要以具有现代版 C++特性和编程元素的代码进行算法描述，因此我们很自然地推荐以 C/C++进行算法实验。当然，原则上进行算法实验可以使用任何程序设计语言，读者尽可选择自己喜欢和熟悉的程序设计语言进行算法实验。本教材选用现代版 C++还有三个独特的理由：一是 C/C++是计算机科学的专业程序设计语言，其对算法机理的揭示能力远强于其他程序设计语言；二是 C/C++是运行效率极高的程序设计语言，这一点对于算法性能测试尤其重要；三是 C++编程系统与传统的 C 语言编程系统完全兼容，这样就可以在不用增加类定义开销的情况下以简洁的代码实现算法，尤其适用于算法的教与学过程。

1.5.1　现代版 C++的算法实验环境建议

“工欲善其事，必先利其器”，这句格言尤其适用于计算机程序设计领域。现代专业编程都是在集成开发环境（Integrated Development Environment，IDE）下进行的，几乎每种程序设计语言都有易学、易用和功能强大的 IDE 支持。对于 C/C++，我们推荐两款 IDE：一款是微软的 Visual Studio（VS）；另一款是 Code::Blocks。下面对它们进行扼要介绍。限于篇幅，本教材不介绍其详细安装过程，读者可参阅网上资源进行安装。

微软的 Visual Studio 为编程提供的强大功能，以及由此带来的极高的编程效率，是其他 IDE 无法匹敌的。如果读者要用 C/C++进行算法实现，强烈建议使用其最新的社区版（免费版）Visual Studio Community 2022，注意在安装时要选择“使用 C++的桌面开发”，安装 Visual C++ 2022 及相关的组件。Visual C++ 2022 处在不断更新的过程中，除功能不断增强以外，还不断增强对最新 C++标准的支持。截至 2022 年 4 月的 17.1.5 版，它已经实现了对绝大部分 C++1x 甚至 C++20 功能的支持。本教材中的所有算法都是在 Visual C++ 2022 上测试通过的，所有算法的源代码都在本教材附录中。

Code::Blocks 是一个开源、免费、跨平台的 C/C++及 Fortran IDE。可以说在 Windows 的开源 C/C++ IDE 中，Code::Blocks 是最好的。Code::Blocks 并不像 Visual C++那样带有一体化设计的内置 C/C++编译器，而需要安装和配置另外的 C/C++编译器。Code::Blocks 默认捆绑了 MinGW，即著名的 Linux 上 Gnu C/C++编译器（gcc）的一个 Windows 移植。最新 Code::Blocks 安装包中的 gcc 支持现代版 C++（包括 C++1x 和 C++20 标准）的几乎所有功能。

这两款 C/C++ IDE 都有强大且丰富的功能，能够使开发者以很高的效率编写专业、健壮、高质量的代码。限于篇幅，本教材不对它们的使用进行详述，读者可参考相关资源进行深入和专业的学习。读者进行算法实验需要特别注意以下两个方面。

1．工程化的编程思维

完成一个编程任务通常需要许多与编程有关的代码文件，对于 C/C++来说，通常会有很多个类型为.h/.hpp 的头文件和类型为.C/.CPP 的源代码文件。现代 IDE 均会将它们组织到一个工程（或项目，即 Project）中，进行工程化管理。Visual Studio 提供了更高层级的解决方案（类型为.sln）管理机制，一个解决方案可包括多个工程，其中 Visual C++的工程文件类型为.vcxproj，Code::Blocks 的工程文件类型为.cbp。注意，工程文件由 IDE 自动维护，编程者只

需要学习 IDE 界面提供的创建和管理维护工程的操作，不需要从较原始的意义上操作工程文件。为便于管理工程中的各类文件，通常将它们组织在一个合理的树形文件夹结构中。工程化的编程思维已经成为现代编程基本且必须的要素，读者应好好学习和掌握。

注：在 IDE 运行过程中会自动生成很多文件，如与工程有关的文件，编译生成的目标码文件（.o 或.obj），帮助调试的源映像文件（.map），以及可执行文件（.exe）等。但编程者的工作多在源代码文件中进行，这一点对于学生来说尤其重要。当以电子文档提交编程实验报告时，大部分时候编写的程序应该只需提交源代码文件，不应该包括.o、.obj、.map、.exe 等类型的、不必要且占用大量传输和存储资源的文件。

2．断点调试技术

没有人能够保证自己编写的程序没有错误，查找和定位程序中的错误并进行改正是编程工作的重要组成部分。这项工作被定义了一个专门的名字——程序调试（Debugging）。断点调试技术是现代程序调试的核心技术，它将特定的程序行设置为断点，使程序执行到该断点后停下来，编程者可以通过查看该断点处的变量、数组或对象的值分析程序的行为，从而找出程序中的错误。Visual Studio 和 Code::Blocks 都提供了强大的 GUI 式断点调试功能，对于程序设计人员来说，掌握这些功能对于提升编程能力至关重要，是成长为专业程序设计人员的必要过程。

此外，还应知晓一些 C++资源站，以便随时查阅，学习准确的语言语法和库函数功能、格式、参数及其使用方法，cplusplus[14]和 cppreference[15]就是两个优秀的 C++资源站。

1.5.2 算法的现代版 C++实现方式——以 Euclid GCD 算法为例

为了帮助读者快速、高效率、专业化地进行算法实验，使算法学习能够实质性地落地，本教材提供了各算法的专业化现代版 C++实现代码。同时，本教材也拿出了一定篇幅对算法的现代专业化实现技术和手法进行介绍。本节就对 Euclid GCD 算法的现代版 C++实现进行介绍，其中的内容不单单针对 Euclid GCD 算法，而是一种基本的算法实验编程模式。

Euclid GCD 算法的现代版 C++实现代码参见附录 1-1。该代码给出了本课程算法代码的基本结构：代码由许多个代码块组成，各代码块中的代码行单独按顺序编号；代码块 1 是现代版 C++的头文件引用、前向的函数及类声明、全局变量及对象定义等；代码块 2 通常为算法的核心函数，在有些情况下代码块 2 为核心函数的调用函数，这时代码块 3 就会是算法的核心函数；后续的代码块为算法的辅助函数；最后一个代码块为算法的测试函数。这个基本结构简洁、专业且具有很好的包容性。实践证明，这种结构适用于本教材中的所有算法。

此外，该代码还具有两个重要特征。

1．使用命名空间防止名字冲突

代码块 1 第 2 行“namespace NS_EuclidGCD {”是 C++的命名空间定义，它对应的结束符“}”在单独的代码块 3 中。使用命名空间防止名字冲突是现代编程的重要技术。为使本教材中所有算法的实现均具有可管理性和可重用性，应该将它们组织在一个编程工程中，这时使用命名空间防止名字冲突就很必要。

2．使用预置数据提高程序测试效率

C 语言初学者往往会使用 scanf 函数从键盘输入数据进行程序测试，C++初学者会使用标

准输入设备 cin，但这是陈旧且极其低效率的方式。Euclid GCD 程序测试模块使用了二维数组预置多组测试数据，这样不仅不用每次测试运行都手工从键盘输入数据，而且可以使每次的测试运行在多组数据上进行。尽管这看起来是一个很微小的技巧，但是程序从编写出来到测试正确要经历很多遍的反复运行，如果每次都从键盘输入数据，那么将带来巨大的时间浪费。对于涉及较大输入数据量的排序及图算法，从键盘输入数据更是不可取也是不可行的方式，预置数据方式更显优势。

1.6　算法国粹——《九章算术》中的二进制 GCD 算法

求解 GCD 其实还有一个比 Euclid GCD 算法有更高效率的算法，那就是二进制 GCD 算法。本节将首先介绍我国古代数学家在成书于约公元 1 世纪的《九章算术》中提出的二进制 GCD 算法，给出其实例计算演示；然后介绍 Josef Stein 于 1967 年发表的现代版的二进制 GCD 算法。

1.6.1　《九章算术》中的更相减损术——最早的二进制 GCD 算法

《九章算术》是中国古代的数学专著[16]，全书总结了战国、秦、汉时期的数学成就，是我国古代算经十书（汉唐之间出现的十部古算书）中最重要的一部典籍。根据研究，西汉的张苍、耿寿昌曾经做过增补。最后成书最迟在东汉前期，如今流传的大多是在三国时期魏元帝景元四年（263 年）刘徽为《九章算术》所作的注本。

《九章算术》奠定了中国古代数学的基本特点，“算法化”就是其中之一。以现代观点来看，《九章算术》中的“术”就是指算法[17]。

《九章算术》共有九章，计 246 道题。每道题先以例子形式表述，然后以“答曰”给出结果，最后给出方法描述。其第一章“方田”中有一道求 91 和 49 两个数约分的题，其方法称为“约分术”。后来的研究表明，该方法也能用来求最大公约数，其被命名为“更相减损术”。D. E. Knuth 在其《计算机程序设计艺术》中认为这个方法是最早的二进制 GCD 算法[18]。

下面给出《九章算术》中的“约分术”描述。

又有九十一分之四十九。问约之得几何?

答曰：十三分之七。

约分术曰：可半者半之，不可半者，副置分母子之数，以少减多，更相减损，求其等也。以等数约。

这里的“约分术”可用现代语言描述如下：分子和分母如果都是偶数，则将它们同时除以 2；当两者有一个不是偶数时，就将它们列成两列（副置），从大数中减去小数（以少减多），此过程循环进行，直至两数相等，这个相等的数就是分子和分母的约数。

注：《九章算术》中的“术”离严谨意义上的现代算法定义，即算法的克努特定义和图灵机定义，还是有一定距离的，如上面过程最后得到的相等数，应该是原始分子和分母同时循环除以 2 至有一个不是偶数时的分子和分母的约数。但考虑到这些方法发明于远在现代算法定义之前的 1800 多年前，可见我们祖先的数学成就堪称人类计算探索史上的奇迹。

下面以表 1-3 和表 1-4 给出更相减损术的两个示例。其中，表 1-3 中给出的正是《九章算术》中“约分术”的例子。由于 91 和 49 都是奇数，因此并不需要执行“可半者半之”的步骤。表 1-4 中的 420 和 756 都是偶数，因此其第 1 步执行的就是“可半者半之”，由于 210 和

378 仍然是偶数，因此需要再执行一次“可半者半之”的步骤。由表 1-4 可以看出，由更相减损术最后得到的结果 21，是在“可半者半之”步骤执行完成后得到的数，即第 3 步中的 105 和 189 的最大公约数。

表 1-3　更相减损术示例一

No	*a*	*b*	*a*−*b*
1	91	49	42
2	49	42	7
3	42	7	35
4	35	7	28
5	28	7	21
6	21	7	14
7	14	7	7
8	7	7	

表 1-4　更相减损术示例二

No	*a*	*b*	*a*−*b*
1	420	756	—
2	210	378	—
3	105	189	84
4	105	84	21
5	84	21	63
6	63	21	42
7	42	21	21
8	21	21	

1.6.2　现代版的二进制 GCD 算法

现代版的二进制 GCD 算法由 Josef Stein 于 1967 年发表[19, 20]，该算法的基本思想如算法 1-6 所示[21]，其所对应的伪代码如算法 1-7 所示。由此可以看出，其核心就是更相减损术中的方法。

算法 1-6　二进制 GCD 算法的思想描述

对于给定的 a 和 b，如果均为偶数，则循环执行 GCD(a,b) = 2 GCD(a/2, b/2)，因为 2 是 GCD 的因子。

此后循环执行如下操作（注意，此后 2 不会是 GCD 的因子）：

如果 a 和 b 中有一个为偶数（至多有一个为偶数），则设其为 a，循环执行 GCD(a,b)＝GCD(a/2, b)。

设 b 为两者中的较小者（此时 a 和 b 均为奇数），循环执行 GCD(a,b)＝GCD((a-b)/2, b)，直至 a-b=0。

算法 1-7　二进制 GCD 算法

输入：正整数 a, b
输出：a, b 的 GCD
1．k = 0
2．while both a and b are even
3．　　a = a/2; b = b/2; k++
4．end while
5．while a > 0
6．　　if a is even then a = a/2
7．　　else if b is even then b = b/2
8．　　else t = |a-b|/2
9．　　if a < b, then b = t else a = t
10．end while
11．return b << k

下面给出二进制 GCD 算法的复杂度分析[22]。算法第 5 步的主循环最多执行 $O(n)$ 次，其中 n 是 a 和 b 中较大者的二进制位数，因为每执行 2 次就会使 a 和 b 之一减半。对任意精度的 a 和 b，每一步中的减法和移位运算要花费 $O(n)$ 的位操作，因此算法的渐近复杂度为 $O(n^2)$，这与 Euclid GCD 算法相同。由于减法和移位运算的代价比求余数运算低，因此可以

期望它是一个比 Euclid GCD 算法运行效率更高的算法。事实上，Akhavi 和 Vallée 的精确分析表明，二进制 GCD 算法的位运算量仅是 Euclid GCD 算法的 60%[23]。

表 1-5 和表 1-6 分别给出了与表 1-3 和表 1-4 相同数据的二进制 GCD 算法示例。由此可以看出，即使在如此小的数据上，也展示出了因引入偶数减半而带来的运行效率提高。

表 1-5　二进制 GCD 算法示例一

No	a	b	运　　算
1	91	49	$k=0$
2	91	49	$\lvert a-b\rvert/2=21$
3	21	49	$\lvert a-b\rvert/2=14$
4	21	14	$b/2=7$
5	7	7	$\lvert a-b\rvert/2=0$
6	0	7	

表 1-6　二进制 GCD 算法示例二

No	a	b	$a-b$
1	420	756	$k=2$
2	105	189	$\lvert a-b\rvert/2=42$
3	105	42	$b/2=21$
4	105	21	$\lvert a-b\rvert/2=42$
5	42	21	$a/2=21$
6	21	21	$\lvert a-b\rvert/2=0$
7	0	21	

习题

1．找出所用编程系统中的底函数和顶函数，并设计一组实验验证其运算结果，说明为什么所设计的实验能够有效地说明底函数和顶函数的功能。

注 1：鉴于本教材习题涉及传统的问题回答、CD-AV 演示练习、编程实验等多种形式非常不同的类型，为提高学生做作业和教师批改作业的效率，建议以规范化的电子版方式提交作业，即每周每个学生建立一个以“周次-学号-姓名”（如 01-999988880123-陈小明）方式命名的作业文件夹，将需要提交的 Word 作业以“周次-学号-姓名.docx”方式命名并存放到作业文件夹下。Word 作业包括的内容：①传统的问题回答，一般手写后拍照粘贴到 Word 文档中，也可根据任课教师安排直接录入；②CD-AV 演示练习记录，一般对于一组输入数据给出有代表性的 2～4 个步骤的截图，如果是交互练习，则应包括有代表性的交互步骤的截图，一般还应包括每组数据最后结果的截图，对于有 I/O 页的 CD-AV 演示，还应该包括一个有代表性的 I/O 页的截图；③编程实验记录，通常包括两部分内容，一部分是关于编程实验的扼要说明，另一部分是运行结果截图。

注 2：编程的源代码文件应该放到作业文件夹下的 program 文件夹中，若编程问题只有一个源代码文件，则可将其直接复制到 program 文件夹中；若编程问题涉及多个源代码文件，则应将其复制到 program 文件夹下一个合理命名的文件夹中。需要特别注意的是，应仅提交与算法有关的源代码文件，既不要包括编译后的.o、.obj、.exe 等类型的目标码文件，也不要包括编程系统的工程文件及库文件，并且源代码文件要原样提供，既不要改名，也不要复制到.txt 类型的文本文档或 Word 文档中。

注 3：所有有关的截图或图片文件都应粘贴到 Word 文档中，而不要直接复制到作业文件夹中，图片应进行适当的缩放以便于阅读；其他文件，如 Excel 文件，则应复制到作业文件夹中。

注 4：最后将作业文件夹打包为.zip 类型的压缩文件，按任课教师要求的时间发给课代表，由课代表统一打包发给任课教师。

2．给出 200 到 210 之间 11 个数的因子分解式，并将结果表示在 Excel 表格中，同时设计 Excel 表格，给出这 11 个数两两之间的 GCD。

3．用 Excel 表格中的计算实现如表 1-1 所示的 Euclid GCD 算法计算过程，并以 1035 与

759、40902 与 24140、432 与 95256 进行算法实验。

4．举出两个著名的以理论方法解决的科学问题。

5．举出两个著名的以实验方法解决的科学问题。

6．给出算法的克努特定义。

7．试列出现代版 C++的编程特征。

8．给出 Euclid GCD 定理，写出 Euclid GCD 算法的伪代码。

9．在 CAAIS 中用两组数据进行 Euclid GCD 算法的演示练习，熟悉 CAAIS 中的帮助信息查看方式、输入数据选择与输入、升序与降序演示方式、代码跟踪等。

10．在 CAAIS 中用两组数据对 Euclid GCD 算法的 CD-AV 演示进行 30%的交互练习，其中一组数据的两个值要求都大于 2000，并保存结果。

11．编程实现 Euclid GCD 算法，要求输出运算过程中每一步的 a、b 和 r 值，并用第 3 题中的三组数进行测试。

参考文献

[1] CORMEN T H，LEISERSON C E，RIVEST R L，etal. 算法导论[M]. 第 3 版. 殷建平，徐云，王刚，等译. 北京：机械工业出版社，2013.

[2] 段会川，王金玲，徐连诚. 计算教学论——从算法之教学启程[J]. 中国计算机学会通讯，2021，17（3）：62-67.

[3] LEVITIN A. 算法设计与分析基础[M]. 第 3 版. 潘彦，译. 北京：清华大学出版社，2015：7.

[4] LIGO-Caltech[EB/OL]. [2020-12-12]. （链接请扫下方二维码）

[5] KNUTH D E. 计算机程序设计艺术，第 1 卷，基本算法[M]. 第 3 版. 苏运霖，译. 北京：国防工业出版社，2002：4-6.

[6] COFFEY B. Happy 80th Birthday to the Turing Machine![EB/OL]. （2016-12-12）[2020-12-12]. （链接请扫下方二维码）

[7] SIPSER M. 计算理论导引[M]. 第 3 版. 段磊，唐常杰，译. 北京：机械工业出版社，2015.

[8] Wikipedia. Turing Machine[EB/OL]. [2020-12-12]. （链接请扫下方二维码）

[9] Encyclopedia of Mathematics. Turing machine[EB/OL]. [2020-12-12]. （链接请扫下方二维码）

[10] Lucidchart. Flowchart Symbols and Notation[EB/OL]. [2020-12-12].（链接请扫下方二维码）

[11] GRIMM R. The Next Big Thing：C++20[EB/OL].（2019-10-18）[2020-12-12].（链接请扫下方二维码）

[12] STROUSTRUP B. C++11 - the new ISO C++ standard[EB/OL]. [2020-12-12].（链接请扫下方二维码）

[13] 段会川，王金玲，徐连诚. CAAIS[EB/OL].（链接请扫下方二维码）

[14] cplusplus[EB/OL]. [2021-1-5].（链接请扫下方二维码）

[15] cppreference[EB/OL]. [2021-1-5].（链接请扫下方二维码）

[16] 百度百科. 九章算术[EB/OL]. [2021-5-6].（链接请扫下方二维码）

[17] 孙宏安.“约分术”与算法[J]. 高中数学教与学，2004，12：44.

[18] KNUTH D E. 计算机程序设计艺术，第 2 卷，半数值算法[M]. 第 3 版. 苏运霖，译. 北

京：国防工业出版社，2002：309.
[19] STEIN J. Computational problems associated with Racah algebra[J]. Journal of Computational Physics，1967，1（3）：397-405.
[20] KNUTH D E. 计算机程序设计艺术，第 2 卷，半数值算法[M]. 第 3 版. 苏运霖，译. 北京：国防工业出版社，2002：306.
[21] BLACK P E. Binary GCD[EB/OL].（2020-11-2）[2021-5-12].（链接请扫下方二维码）
[22] Wikipedia. Binary GCD algorithm[EB/OL]. [2021-5-12].（链接请扫下方二维码）
[23] AKHAVI A，Vallée B. Average Bit-Complexity of Euclidean Algorithms 1853[C]. Proceedings ICALP'00，Lecture Notes in Computer Science. 2000：373-387.

第 1 章
参考文献链接