职业技术教育软件人才培养模式改革项目成果教材

数据结构与算法

王晓东 编

高等教育出版社

职业技术教育软件人才培养模式改革项目成果教材

数据结构与算法

王晓东　编

高等教育出版社

内容提要

本书是职业技术教育软件人才培养模式改革项目成果教材之一，主要内容包括数据结构和算法的基本概念如表、栈、队列、递归、排序与选择、树、集合、符号表、字典、优先队列、并查集和图等。

为了适应培养我国21世纪计算机各类人才的需要，结合我国高等学校教育工作的现状，立足培养学生能跟上国际计算机科学技术的发展水平，更新教学内容和教学方法，本书以基本数据结构和算法设计策略为知识单元系统地介绍数据结构知识与应用、计算机算法的设计与分析方法，为计算机学科的学生提供一个广泛坚实的数据结构与算法设计基础知识。

本书适用于高等职业学校、高等专科学校、成人高校、独立设置的软件职业技术学院、本科院校及举办的二级职业技术学院、教育学院以及民办高校使用，不仅可用作高等院校计算机科学与工程专业学生学习数据结构与算法的教材，而且也适合广大工程技术人员和自学读者学习参考。

图书在版编目(CIP)数据

数据结构与算法/王晓东编 .—北京：高等教育出版社；2003.12（2005重印）

ISBN 7-04-013204-4

Ⅰ.数… Ⅱ.王… Ⅲ.①数据结构-教材②算法分析-教材 Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2003）第098331号

出版发行	高等教育出版社	购书热线	010-58581118
社　　址	北京市西城区德外大街4号	免费咨询	800-810-0598
邮政编码	100011	网　　址	http://www.hep.edu.cn
总　　机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经　　销	北京蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印　　刷	北京机工印刷厂		
开　　本	787×1092　1/16	版　　次	2003年12月第1版
印　　张	17.25	印　　次	2005年5月第3次印刷
字　　数	420 000	定　　价	21.80元

物料号　13204-00

职业技术教育软件人才培养模式改革项目
成果教材编审委员会

出 版 说 明

信息产业是国民经济和社会发展基础性、战略性产业。加快发展信息技术和信息产业，以信息化带动工业化，以信息化促进工业化，是当前和今后我国产业结构调整发展的战略重点。软件产业是信息产业的核心，加快软件人才培养是加快软件产业发展的先决条件。为适应经济结构战略性调整及软件产业发展的需要，加快培养各类软件应用性人才，在国家改革和发展委员会、教育部的指导和支持下，福建省从2002年开始，在全国率先举办软件类高等职业技术教育，拟以办学模式和人才培养模式改革为重点，积极探索有水平、有质量、有特色的软件高职教育发展的新路子。

在软件类高等职业技术教育改革和建设过程中，福建省坚持教育创新，把改革教学内容和课程体系，加强专业建设、教材建设和教学队伍建设作为工作的重点。目前，根据软件行业发展趋势、就业环境和软件高等职业技术教育的办学特点，经组织专家论证和审定，福建省高校首批开设了可视化编程、Web应用程序设计、软件测试、网络系统管理员、网络构建技术、数据库管理员、图形/图像制作、多媒体制作、计算机办公应用等9个软件高职专业，制订了较为科学合理的人才培养方案。为配合支持软件类高职教育的改革和建设，福建省教育厅聘请软件教育有关专家、学者和著名软件企业的高级工程技术人员成立了“职业技术教育软件人才培养模式改革项目成果教材编审委员会”，以“抓好试点规划，实施精品战略”为指导方针，认真吸取国内外软件技术发展成果，根据软件企业对人才培养提出的新要求和软件高职的办学特点，认真处理好教材的统一性与多样化、基本教材与辅助教材、学历教育教材与认证培训教材的关系，以组织开展软件高职公共基础课、专业基础课和专业主干课教材的建设为重点，同时扩大品种，实现教材系列配套，在此基础上形成特色鲜明、优化配套的软件高等职业技术教育教材体系。

本软件系列教材适用于本科院校、高职高专院校、成人高校及继续教育学院的软件高职类专业及相关专业使用。

职业技术教育软件人才培养模式改革项目成果教材编审委员会

二〇〇三年五月

前　言

为了以最少的成本、最快的速度、最好的质量开发出适合各种应用需求的软件，软件设计人员在软件开发过程中必须遵循软件工程的原则。一个高效的程序不仅需要“编程小技巧”，它更需要合理的数据组织和清晰高效的算法，这正是计算机科学领域里数据结构与算法设计所研究的主要内容。一些著名的计算机科学家在有关计算机科学教育的论述中认为，计算机科学教育是一种创造性思维活动，其教育必须面向设计。数据结构与算法正是一门面向设计，且处于计算机学科核心地位的教育课程。通过对数据结构与算法的系统学习与研究，理解和掌握算法设计的主要方法，培养对算法的计算复杂性进行正确分析的能力，对从事计算机系统结构、系统软件和应用软件研究与开发的科技工作者是非常重要和必不可少的。为了适应培养我国21世纪计算机各类人才的需要，结合我国高等学校教育工作的现状，立足培养学生跟上国际计算机科学技术的发展水平，更新教学内容和教学方法，本书以基本数据结构和算法设计策略为知识单元系统地介绍数据结构知识与应用、计算机算法的设计与分析方法，以期为计算机学科的学生提供一个广泛坚实的数据结构与算法设计基础知识。

全书共分13章，首先在第1章中介绍了数据结构、抽象数据类型和算法的基本概念，接着对算法的计算复杂性和算法的描述作了简要的阐述。然后以抽象数据类型为主线索，围绕设计算法常用的基本数据结构和基本设计策略组织了第2章～第13章的内容。

第2章～第4章依次介绍基于序列的抽象数据类型表、栈和队列。

第5章介绍了递归的概念，以及递归在数据结构和算法设计中的广泛应用，并详细阐述了分治法、动态规划和回溯法这3个实践中常用的使用递归技术的算法设计策略。

第6章介绍在实际应用中常用的排序与选择算法。

第7章讨论反映层次关系的抽象数据类型树。

第8章讨论表示集合的抽象数据类型。

第9章讨论抽象数据类型符号表以及散列表等实践中常用实现符号表的方法。

第10章的主题是以有序集为基础的抽象数据类型字典及其实现方法。讨论了实现字典的二叉搜索树以及AVL树等高效算法。

第11章讨论以集合为基础的抽象数据类型优先队列及其实现方法。

第12章讨论以不相交的集合为基础的抽象数据类型并查集及其实现方法。

最后，在第13章介绍非线性结构图及图的算法。

为了加深对知识的理解，各章配有难易适当的习题，以适应不同程度读者练习的需要。

由于作者的知识和写作水平有限，书稿虽几经修改，仍难免有缺点和错误。热忱欢迎同行专

家和读者的批评指正，以使本书在使用中不断改进，日臻完善。

在本书的编写过程中，福州大学“211 工程”计算机与信息工程重点学科实验室为本书的写作提供了优良的设备与工作环境。傅清祥教授在百忙之中认真审阅了全书，提出了许多宝贵的改进意见。在此，谨向每一位曾经关心和支持本书编写工作的各方面人士表示衷心的谢意！

作者

2003 年 6 月

目　录

第1章 引 论

学习目标

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 能够列举求解问题的基本步骤。
- 掌握算法在最坏情况、最好情况和平均情况下的计算复杂性概念。
- 掌握算法复杂性的渐近性态的数学表述。
- 了解表达算法的抽象机制。
- 熟悉抽象数据类型的基本概念。
- 熟悉数据类型和数据结构的概念。
- 理解数据结构、数据类型和抽象数据类型三者的区别和联系。
- 掌握用C语言描述数据结构与算法的方法。

1.1 算法及其复杂性的概念

1.1.1 算法与程序

对于计算机科学来说，算法(algorithm)的概念是至关重要的。例如在一个大型软件系统的开发中，设计出有效的算法将起决定性的作用。通俗地讲，算法是指解决问题的一种方法或一个过程。严格地讲，算法是由若干条指令组成的有穷序列，且满足下述几条性质：

① 输入：有零个或多个由外部提供的量作为算法的输入。

② 输出：算法产生至少一个量作为输出。

③ 确定性：组成算法的每条指令是清晰的、无歧义的。

④ 有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序(program)与算法不同。程序是算法用某种程序设计语言的具体实现,程序可以不满足算法的性质(4)。例如操作系统,它是一个在无限循环中执行的程序,因而不是一个算法,然而可把操作系统的各种任务看成是一些单独的问题,每一个问题由操作系统中的一个子程序通过特定的算法来实现,该子程序得到输出结果后便终止。

1.1.2 算法复杂性的概念

一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上,所需要的资源越多,就说该算法的复杂性越高;反之,所需要的资源越少,就说该算法的复杂性越低。计算机的资源,最重要的是时间和空间(即存储器)资源。因此,算法的复杂性有时间复杂性和空间复杂性之分。不言而喻,对于任意给定的问题,设计出复杂性尽可能低的算法是设计算法时追求的一个重要目标。另一方面,当给定的问题已有多种算法时,选择其中复杂性最低者,是选用算法应遵循的一个重要准则。因此,算法复杂性分析对算法的设计或选用有重要的指导意义和实用价值。确切地说,算法的复杂性是运行算法所需要的计算机资源的量。需要的时间资源的量称为时间复杂性;需要的空间资源的量称为空间复杂性。这个量应该集中反映算法的效率,而从运行该算法的实际计算机中抽象出来,换句话说,这个量应该是只依赖于算法要解的问题的规模和算法的输入函数。如果分别用 n 和 I 表示算法要解的问题的规模和算法的输入,而且用 C 表示复杂性,那么,应该将算法复杂性表示为 $C(n,I)$。如果把时间复杂性和空间复杂性分开,并分别用 T 和 S 来表示,那么应该有:$T=T(n,I)$ 和 $S=S(n,I)$。由于时间复杂性与空间复杂性概念类同,计量方法相似,且空间复杂性分析相对简单些,所以本书将主要讨论时间复杂性。现在的问题是如何将复杂性函数具体化,即对于给定的 n 和 I,如何导出 $T(n,I)$和 $S(n,I)$的数学表达式,给出计算 $T(n,I)$和 $S(n,I)$的法则。下面以 $T(n,I)$为例,将复杂性函数具体化。

根据 $T(n,I)$的概念,它应该是算法在一台抽象的计算机上运行所需要的时间。设此抽象的计算机所提供的元运算有 k 种,分别记为 $O_1,O_2,\cdots,O_k$。又设每执行一次这些元运算所需要的时间分别为 $t_1,t_2,\cdots,t_k$。对于给定的算法 A,设经统计,用到元运算 O_i 的次数为 $e_i,i=1,2,\cdots,k$。很清楚,对于每一个 $i,1\leqslant i\leqslant k$,$e_i$ 是 n 和 I 的函数,即 $e_i=e_i(n,I)$。那么有 $T(n,I)=\sum_{i=1}^{k}t_ie_i(n,I)$,其中 $t_i,i=1,2,\cdots,k$,是与 n 和 I 无关的常数。

显然,不可能对规模为 n 的每一种合法的输入 I 都统计 $e_i(n,I),i=1,2,\cdots,k$。因此 $T(n,I)$的表达式还得进一步简化,或者说,只能在规模为 n 的某些或某类有代表性的合法输入中统计相应的 $e_i,i=1,2,\cdots,k$,并评价时间复杂性。

通常考虑 3 种情况下的时间复杂性,即最坏情况、最好情况和平均情况下的时间复杂性,并分别记为 $T_{\max}(n)$、$T_{\min}(n)$和 $T_{\text{avg}}(n)$。在数学上有

$$T_{\max}(n)=\max_{I\in D_n}T(n,I)=\max_{I\in D_n}\sum_{i=1}^{k}t_ie_i(n,I)=\sum_{i=1}^{k}t_ie_i(n,I^*)=T(n,I^*)$$

$$T_{\min}(n)=\min_{I\in D_n}T(n,I)=\min_{I\in D_n}\sum_{i=1}^{k}t_ie_i(n,I)=\sum_{i=1}^{k}t_ie_i(n,\tilde{I})=T(n,\tilde{I})$$

$$T_{\text{avg}}(n)=\sum_{I\in D_n}P(I)T(n,I)=\sum_{I\in D_n}P(I)\sum_{i=1}^{k}t_ie_i(n,I)$$

其中，D_n 是规模为 n 的合法输入的集合；I^* 是 D_n 中一个使 $T(n,I^*)$ 达到 $T_{max}(n)$ 的合法输入；$\tilde{I}$ 是 D_n 中一个使 $T(n,\tilde{I})$ 达到 $T_{min}(n)$ 的合法输入；而 $P(I)$ 是在算法的应用中出现输入 I 的概率。

以上 3 种情况下的时间复杂性各从某一个角度来反映算法的效率，各有各的局限性，也各有各的用处。实践表明，可操作性最好且最有实际价值的是最坏情况下的时间复杂性。本书对算法时间复杂性的分析主要采用最坏情况下时间复杂性分析。

1.1.3　算法复杂性的渐近性态

随着经济的发展、社会的进步、科学研究的深入，要求用计算机解决的问题越来越复杂，规模越来越大。对求解这类问题的算法作复杂性分析具有特别重要的意义，因而要特别关注。为此，要引入复杂性渐近性态的概念。设 $T(n)$ 是前面所定义的关于算法 A 的复杂性函数。一般说来，当 n 单调增加且趋于∞时，$T(n)$ 也将单调增加趋于∞。对于 $T(n)$，如果存在 $\tilde{T}(n)$，使得当 $n\to\infty$ 时有 $\dfrac{T(n)-\tilde{T}(n)}{T(n)}\to 0$，那么，就说 $\tilde{T}(n)$ 是 $T(n)$ 当 $n\to\infty$ 时的渐近性态，或称 $\tilde{T}(n)$ 为算法 A 当 $n\to\infty$ 的渐近复杂性而与 $T(n)$ 相区别。因为在数学上，$\tilde{T}(n)$ 是 $T(n)$ 当 $n\to\infty$ 时的渐近表达式。直观上，$\tilde{T}(n)$ 是 $T(n)$ 中略去低阶项所留下的主项。所以它无疑比 $T(n)$ 来得简单。例如当 $T(n)=3n^2+4n\log n+7$[①] 时 $\tilde{T}(n)$ 的一个答案是 $3n^2$，因为这时有

$$\lim_{n\to\infty}\frac{T(n)-\tilde{T}(n)}{T(n)}=\lim_{n\to\infty}\frac{4n\log n+7}{3n^2+4n\log n+7}=0$$

显然 $3n^2$ 比 $3n^2+4n\log n+7$ 简单得多。

由于当 $n\to\infty$ 时 $T(n)$ 渐近于 $\tilde{T}(n)$，有理由用 $\tilde{T}(n)$ 来替代 $T(n)$ 作为算法 A 在 $n\to\infty$ 时的复杂性的度量。而且由于 $\tilde{T}(n)$ 明显比 $T(n)$ 简单，这种替代是对复杂性分析的一种简化。进一步，考虑到分析算法的复杂性的目的在于比较求解同一问题的 2 个不同算法的效率。而当要比较的 2 个算法的渐近复杂性的阶不相同时，只要能确定出各自的阶，就可以判定哪一个算法的效率高。换句话说，这时的渐近复杂性分析只要关心 $\tilde{T}(n)$ 的阶就够了，不必关心包含在 $\tilde{T}(n)$ 中的常数因子。所以，常常又对 $\tilde{T}(n)$ 的分析进一步简化，即假设算法中用到的所有不同的元运算各执行一次所需要的时间都是一个单位时间。

综上所述，已经给出了简化算法复杂性分析的方法，即只要考察当问题的规模充分大时，算法复杂性在渐近意义下的阶。本书的算法分析都将这么做。与此简化的复杂性分析相配套，需要引入以下渐近复杂性的记号。

以下设 $f(n)$ 和 $g(n)$ 是定义在正数集上的正函数。

如果存在正的常数 c 和自然数 n_0，使得当 $n\geqslant n_0$ 时有 $f(n)\leqslant cg(n)$，则称函数 $f(n)$ 当 n 充分大时上有界，且 $g(n)$ 是它的一个上界，记为 $f(n)=O(g(n))$。这时还说 $f(n)$ 的阶不高于 $g(n)$ 的阶。

举几个例子：

① 因为对所有的 $n\geqslant 1$ 有 $3n\leqslant 4n$，从而有 $3n=O(n)$。

---

① 本书 $\log n$ 表示以 2 为底的 n 的对数。

② 因为当 $n \geqslant 1$ 时有 $n+1\ 024 \leqslant 1\ 025n$，从而有 $n+1\ 024 = O(n)$。

③ 因为当 $n \geqslant 10$ 时有 $2n^2+11n-10 \leqslant 3n^2$，从而有 $2n^2+11n-10 = O(n^2)$。

④ 因为对所有 $n \geqslant 1$ 有 $n^2 \leqslant n^3$，从而有 $n^2 = O(n^3)$。

⑤ 作为一个反例 $n^3 \neq O(n^2)$。因为若不然，则存在正的常数 c 和自然数 n_0，使得当 $n \geqslant n_0$ 时有 $n^3 \leqslant cn^2$，即 $n \leqslant c$。显然，当取 $n = \max\{n_0, \lfloor c \rfloor + 1\}$ 时这个不等式不成立，所以 $n^3 \neq O(n^2)$。

按照符号 O 的定义，容易证明它有如下运算规则：

① $O(f)+O(g) = O(\max(f,g))$。

② $O(f)+O(g) = O(f+g)$。

③ $O(f)O(g) = O(fg)$。

④ 如果 $g(n) = O(f(n))$，则 $O(f)+O(g) = O(f)$。

⑤ $O(cf(n)) = O(f(n))$，其中 c 是一个正的常数。

⑥ $f = O(f)$。

规则①的证明：设 $F(n) = O(f)$。根据符号 O 的定义，存在正常数 c_1 和自然数 n_1，使得对所有的 $n \geqslant n_1$，有 $F(n) \leqslant c_1 f(n)$。类似地，设 $G(n) = O(g)$，则存在正的常数 c_2 和自然数 n_2，使得对所有的 $n \geqslant n_2$ 有 $G(n) \leqslant c_2 g(n)$。

令 $c_3 = \max\{c_1, c_2\}$，$n_3 = \max\{n_1, n_2\}$，$h(n) = \max\{f, g\}$，则对所有的 $n \geqslant n_3$，有

$$F(n) \leqslant c_1 f(n) \leqslant c_1 h(n) \leqslant c_3 h(n)$$

类似地，有

$$G(n) \leqslant c_2 f(n) \leqslant c_2 h(n) \leqslant c_3 h(n)$$

因而

$$\begin{aligned} O(f)+O(g) &= F(n)+G(n) \leqslant c_3 h(n)+c_3 h(n) \\ &= 2c_3 h(n) = O(h) = O(\max(f,g)) \end{aligned}$$

其余规则的证明类似，可作为读者的练习。

应该指出，根据符号 O 的定义，用它评估算法的复杂性，得到的只是当规模充分大时的一个上界。这个上界的阶越低则评估就越精确，结果就越有价值。

与渐近复杂性有关的另一记号是 Ω，其定义如下：如果存在正的常数 c 和自然数 n_0，使得当 $n \geqslant n_0$ 时有 $f(n) \geqslant cg(n)$，则称函数 $f(n)$ 当 n 充分大时下有界，且 $g(n)$ 是它的一个下界，记为 $f(n) = \Omega(g(n))$。这时还说 $f(n)$ 的阶不低于 $g(n)$ 的阶。

用 Ω 评估算法的复杂性，得到的只是该复杂性的一个下界。这个下界的阶越高，则评估就越精确，结果就越有价值。这里的 Ω 只对问题的一个算法而言。如果它是对一个问题的所有算法或某类算法而言，即对于一个问题和任意给定的充分大的规模 n，下界在该问题的所有算法或某类算法的复杂性中取值，那么它将更有意义。这时得到的相应下界，称之为问题的下界或某类算法的下界。它常常与符号 O 配合以证明某问题的一个特定算法是该问题的最优算法或该问题的某算法类中的最优算法。

明白了符号 O 和 Ω 之后，符号 θ 也随之明确。定义 $f(n) = \theta(g(n))$ 当且仅当 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$。这时，说 $f(n)$ 与 $g(n)$ 同阶。

最后，如果对于任意给定的 $\varepsilon > 0$，都存在正整数 n_0，使得当 $n \geqslant n_0$ 时有 $f(n)/g(n) < \varepsilon$，则称

函数 $f(n)$当 n 充分大时的阶比 $g(n)$低，记为 $f(n) = O(g(n))$。

例如：$4n\log n + 7 = O(3n^2 + 4n\log n + 7)$。

1.2 算法的表达与数据表示

1.2.1 问题求解

用计算机解决一个稍复杂的实际问题，大体都要经历如下的步骤：

① 将实际问题数学化，即把实际问题抽象为一个带有一般性的数学问题。这一步要引入一些数学概念，精确地阐述数学问题，弄清问题的已知条件和所要求的结果，以及在已知条件和所要求的结果之间存在着的隐式或显式的联系。

② 对于确定的数学问题，设计求其解的方法，即所谓的算法设计。这一步要建立问题的求解模型，即确定问题的数据模型并在此模型上定义一组运算，然后借助于对这组运算的执行和控制，从已知数据出发导向所要求的结果，形成算法并用自然语言来表述。这种语言不是程序设计语言，不能被计算机所接受。

③ 用计算机上的一种程序设计语言来表达已设计好的算法。换句话说，将非形式自然语言表达的算法转变为用一种程序设计语言表达的算法。这一步称为程序设计或程序编制。

④ 在计算机上编辑、调试和测试编制好的程序，直到输出所要求的结果。

上述问题求解的过程中，求解问题的算法及其实现是核心内容。本章着重考虑第②步，而且把注意力集中在算法表达的抽象机制上，目的是引入一个重要的概念，即抽象数据类型的概念，同时为大型程序设计提供一种相应的自顶向下逐步求精的模块化方法，即运用抽象数据类型来描述程序的方法。

1.2.2 表达算法的抽象机制

算法是一个运算序列。这个运算序列中的所有运算定义在一类特定的数学模型上，并以解决一类特定问题为目标。这个运算序列应该具备下列 4 个特征：

① 有限性，即序列的项数有限，且每一项运算都在有限时间内完成。

② 确定性，即序列的每一项运算都有明确的定义，无歧义。

③ 可以没有输入运算项，但一定有输出运算项。

④ 可行性，即对于任意给定的合法的输入都能得到相应的正确输出。

这些特征可以用来判别一个确定的运算序列是否称得上是一个算法。

算法的程序表达，归根到底是算法要素的程序表达，因为一旦算法的每一项要素都用程序清楚地表达，整个算法的程序表达也就不成问题。

很明显，算法有如下 3 要素：

① 作为运算序列中各种运算的运算对象和运算结果的数据。

② 运算序列中的各种运算。

③ 运算序列中的控制转移。

这 3 要素依序分别简称为数据、运算和控制。

由于算法层出不穷，变化万千，其中的运算所作用的对象数据和所得到的结果数据名目繁多，不胜枚举。最简单最基本的有布尔值数据、字符数据、整数和实数数据等；稍复杂的有向量、矩阵、记录等数据；更复杂的有集合、树和图，还有声音、图形、图像等数据。

同样由于算法层出不穷，变化万千，其中的运算的种类五花八门、多姿多彩。最基本和最初等的有赋值运算、算术运算、逻辑运算和关系运算等；稍复杂的有算术表达式和逻辑表达式等；更复杂的有函数值计算、向量运算、矩阵运算、集合运算，以及表、栈、队列、树和图的运算等；此外，还可能有以上列举的运算的复合和嵌套。

控制转移相对单纯。在串行计算中，它只有顺序、分支、循环、递归和无条件转移等几种。

最早的程序设计语言是机器语言，即具体的计算机上的一个指令集。当时，要在计算机上运行的所有算法都必须直接用机器语言来表达，计算机才能接受。算法的运算序列包括运算对象和运算结果都必须转换为指令序列。其中的每一条指令都以编码（指令码和地址码）的形式出现。这与用高级程序设计语言表达的算法相差甚远。对于没受过程序设计专门训练的人来说，程序可读性极差。用机器语言表达算法的运算、数据和控制十分繁杂琐碎，因为机器语言所提供的指令太初等、原始。机器语言只接受算术运算、按位逻辑运算和数的大小比较运算等。对于稍复杂的运算，都必须分解到最初等的运算才能用相应的指令替代之。机器语言能直接表达的数据只有最原始位、字节和字 3 种。算法中即使是最简单的数据如布尔值、字符、整数和实数，也必须映射到位、字节和字中，还要分配它们的存储。对于算法中有结构的数据的表达则要麻烦得多。机器语言所提供的控制转移指令也只有无条件转移、条件转移、进入子程序和从子程序返回等最基本的几种。用它们来构造循环、形成分支、调用函数都得事先做许多准备，还需要许多经验和技巧。

直接用机器语言表达算法有许多缺点：

① 大量繁杂琐碎的细节牵制着程序员，使他们不可能有更多的时间和精力去从事创造性的劳动，执行对他们来说是更为重要的任务，如确保程序的正确性、高效性。

② 程序员既要驾驭程序设计的全局又要深入每一个局部直到实现的细节，即使智力超群的程序员也常常会顾此失彼，屡出差错，因而所编出的程序可靠性差，且开发周期长。

③ 由于用机器语言进行程序设计的思维和表达方式与人们的习惯大相径庭，只有经过较长时间职业训练的程序员才能胜任，使得程序设计曲高和寡。

④ 它的书面形式全是“密码”，可读性差，不便于交流与合作。

⑤ 它严重地依赖于具体的计算机，可移植性和可重用性差。

克服上述缺点的出路在于程序设计语言的抽象，让它尽可能地接近于算法语言。为此，人们首先注意到的是可读性和可移植性，因为它们相对地容易通过抽象而得到改善。

汇编语言实现了对机器语言的抽象，它将机器语言的每一条指令符号化：指令码代之以记忆符号，地址码代之以符号地址，使得其含义显现在符号上而不再隐藏在编码中。另一方面，汇编语言摆脱了具体计算机的限制，可在具有不同指令集的计算机上运行，只要该计算机配备了汇编语言的一个汇编程序。这无疑是机器语言朝算法语言靠拢迈出的一步。但是，它离算法语言还太远，以致程序员还不能从分解算法的数据、运算和控制，直至细化到汇编可直接表达的指令等

繁杂琐碎的事务中解脱出来。

高级程序设计语言的出现使算法的程序表达产生了一次飞跃。诚然，算法最终要表达为具体计算机上的机器语言才能在该计算机上运行，得到所需要的结果。但汇编语言的实践启发人们，表达成机器语言不必一步到位，可以分两步走或者可以筑桥过河。即先表达成一种中间语言，然后转成机器语言。汇编语言作为一种中间语言，并没有获得很大成功。原因是它离算法语言还太远。这便指引人们去设计一种尽量接受算法语言的规范语言，即所谓的高级程序设计语言。让程序员可以方便地用它表达算法，然后借助于规范的高级语言到规范的机器语言的“翻译”，最终将算法表达为机器语言。而且，由于高级语言和机器语言都具有规范性，这里的“翻译”完全可以机械化地由计算机来完成，就像汇编语言被翻译成机器语言一样，只要计算机配备一个编译程序。上述两步，前一步由程序员去完成，后一步可以由编译程序去完成。在规定清楚它们各自该做什么之后，这两步是完全独立的。它们各自该如何做则互不相干。前一步要做的只是用高级语言正确地表达给定的算法，产生一个高级语言程序，后一步要做的只是将第一步得到的高级语言程序翻译成机器语言程序。至于程序员如何用高级语言表达算法和编译程序如何将高级语言表达的算法翻译成机器语言表达的算法，显然毫不相干。处理从算法语言最终表达成机器语言这一复杂过程的上述思想方法就是一种抽象。汇编语言和高级语言的出现都是这种抽象的范例。与汇编语言相比，高级语言的巨大成功在于它在数据、运算和控制3方面的表达中引入许多使之十分接近算法语言的概念和工具，大大地提高抽象地表达算法的能力。在运算方面，高级语言除允许原封不动地运用算法语言的算术运算、逻辑运算、关系运算、算术表达式、逻辑表达式外，还引入强有力的函数等工具，并让用户自定义。这一工具的重要性不仅在于它精简了重复的程序文本段，而且在于它反映出程序的二级抽象。在函数调用级，人们只关心它能做什么，不必关心它如何做。只是到定义函数时，人们才给出如何做的细节。用过高级语言的读者都知道，一旦函数的名称、参数和功能被规定清楚，在程序中调用它们便与在程序的头部说明它们完全分开。可以修改一个函数甚至更换函数体而不影响调用该函数。如果把函数名看成是运算名，把参数看成是运算的对象或运算的结果，那么，函数调用和初等运算的引用就完全一样。利用函数以及函数的复合或嵌套可以很自然地表达算法语言中任何复杂的运算。

在数据表示方面，高级语言引入了数据类型的概念，即把所有的数据加以分类。每一个数据（包括表达式）或每一个数据变量都属于其中确定的一类。这一类数据称为一个数据类型。数据类型是数据或数据变量类属的说明，它指示该数据或数据变量可能取的值的全体。对于无结构的数据，高级语言除提供标准的基本数据类型外，还提供用户可自定义的枚举类型、子界类型和指针类型等。这些类型的使用方式都顺应人们在算法语言中使用的习惯。对于有结构的数据，高级语言提供了数组、记录、集合和文件等标准的结构数据类型。其中，数组是科学计算中的向量、矩阵的抽象；记录是商业和管理中的记录的抽象；集合是数学中小集合的势集的抽象；文件是诸如磁盘等外存储数据的抽象。人们可以利用所提供的基本数据类型，按数组、记录、集合和文件的构造规则构造有结构的数据。此外，还允许用户利用标准的结构数据类型，通过复合或嵌套构造更复杂更高层的结构数据。这使得高级语言中的数据类型呈明显的分层。高级语言中数据类型的分层是没有穷尽的，因而用它们可以表达算法语言中任何复杂层次的数据。

在控制方面，高级语言通常提供表达算法控制转移的如下方式：

① 缺省的顺序控制。