普通高等教育
软件工程“十三五”规划教材

13th Five-Year Plan Textbooks
of Software Engineering

工业和信息化普通高等教育
“十三五”规划教材

算法设计与分析

王幸民 张晓霞 ◎ 主编
闫鹏飞 杨崇艳 薛晋东 ◎ 副主编

Design and Analysis of Computer Alogrithms

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

普通高等教育
软件工程
“十三五”规划教材

13th Five-Year Plan Textbooks
of Software Engineering

工业和信息化普通高等教育
“十三五”规划教材

算法设计与分析

王幸民 张晓霞 ◎ 主编
闫鹏飞 杨崇艳 薛晋东 ◎ 副主编

Design and Analysis of Algorithms

人民邮电出版社
北京

图书在版编目（CIP）数据

算法设计与分析 / 王幸民，张晓霞主编. — 北京 : 人民邮电出版社，2018.1
普通高等教育软件工程“十三五”规划教材
ISBN 978-7-115-47266-3

Ⅰ. ①算… Ⅱ. ①王… ②张… Ⅲ. ①电子计算机－算法设计－高等学校－教材②电子计算机－算法分析－高等学校－教材 Ⅳ. ①TP301.6

中国版本图书馆CIP数据核字(2017)第287497号

内 容 提 要

本书以程序设计为基础，数据结构为工具，六大核心算法为目标，系统地介绍了算法设计中典型问题的求解过程。

全书内容包括算法设计与分析基础、递归算法、分治算法、贪心算法、动态规划算法、回溯算法、分支限界算法、实验指导。六大核心算法后都配有典型问题的 C++代码实现，并结合实验指导辅助读者进行算法实践训练。

本书结构清晰，内容翔实，通俗易懂，深入浅出，兼顾理论和实践，可作为高等学校计算机或软件工程专业学生的教材，也可作为工程技术人员的参考书。

◆ 主　　编　王幸民　张晓霞
副 主 编　闫鹏飞　杨崇艳　薛晋东
责任编辑　邹文波
责任印制　沈　蓉　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：13.25　　2018 年 1 月第 1 版
字数：335 千字　　2018 年 1 月北京第 1 次印刷

定价：45.00 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前言 PREFACE

“算法设计与分析”是计算机科学的核心问题，该课程已经成为计算机科学与技术专业及软件工程专业课程体系中一门重要的必修课。本课程目的是通过对计算机领域的许多常见问题和有代表性算法的学习和研究，使读者了解并掌握算法设计的一些主要方法，提高分析问题的基本技能，达到独立设计算法和分析其复杂度的水平。

全书共 7 章。第 1 章介绍算法的基本概念，并对分析算法复杂度的准则及本书用到的基本数据结构的知识做了简要的介绍。第 2 章至第 7 章分别介绍常用的算法设计方法，它们分别是递归算法、分治算法、贪心算法、动态规划算法、回溯算法和分支限界算法。本书选取具有代表性的问题，讲解这些经典算法的思路。另外，为了便于读者掌握各类基本算法，书后附有算法应用的练习。

本书主要体现了如下特点。

（1）以算法设计为主线来组织素材，针对具体问题类，深入分析了算法设计思路、设计步骤、算法描述及算法复杂度等；从问题建模、算法设计与分析、算法改进等方面给出适当的描述，在理论上为实际问题的算法设计与分析提供了清晰、整体的思路和方法。与常见的算法和数据结构教材有所不同，本书并没有过多关注细节，算法描述采用伪码，力求突出对问题本身的分析和求解方法的阐述。

（2）在本书内容的组织上，不仅围绕问题类展开，将不同方法用于求解同一问题并进行讲解，便于读者把握问题分析的发展脉络，还通过比较分布在不同章节、求解同一问题的不同算法，让读者了解算法的设计过程。同时，在各章中还将同一算法的设计方法和设计策略用于不同问题的求解，更便于读者体会、掌握算法设计的思路。

（3）本书的素材来自作者多年的教学积淀，选材适当，组织合理，先引入基本概念和数学基础知识，然后进行算法设计与分析的核心内容讲解。在叙述中不但注意理论的严谨，也精选了大量生动有趣的例子，每章都配有难度适当的练习，适合教学使用。

本书框架由多位教学一线的老师共同讨论制定。参与编写的老师分工如下：杨崇艳编写第 1 章，张晓霞编写第 2 章、第 5 章，王幸民编写第 3 章、第 4 章，闫鹏飞编写第 6 章和实验指导，薛晋东编写第 7 章。

在编写的过程中参考了国内外多种版本的算法设计与分析以及计算复杂性方面的教材、论文和专著，从中吸取了一些优秀的思路和素材，在此一并向有关作者致谢。

编　者

2018 年 1 月

目 录 CONTENTS

第 1 章 算法设计与分析基础 1

1.1 算法概述 2
1.1.1 什么是算法 2
1.1.2 学习算法的重要性 6
1.2 问题的求解过程 6
1.2.1 问题及问题的求解过程 6
1.2.2 算法设计与算法表示 7
1.2.3 算法确认和算法分析 8
1.3 算法的复杂性分析 8
1.3.1 算法评价的基本原则 9
1.3.2 影响程序运行时间的因素 10
1.3.3 算法复杂度 11
1.3.4 使用程序步分析算法 14
1.3.5 渐近表示法 15
1.4 算法设计中常见的重要问题类型 18
1.4.1 排序问题 18
1.4.2 查找问题 19
1.4.3 图问题 19
1.4.4 组合问题 20
1.4.5 几何问题 20
1.4.6 数值问题 21
1.4.7 其他常见问题 21
1.5 常用的算法设计方法 22
1.5.1 数值计算算法 23
1.5.2 非数值计算算法 24
1.6 小结 28
练习题 29

第 2 章 递归算法 31

2.1 递归算法的思想 32
2.1.1 递归算法的特性 32
2.1.2 递归算法的执行过程 32
2.1.3 递推关系 33
2.2 递归法应用举例 37
2.2.1 汉诺塔问题 37
2.2.2 斐波那契数列问题 39
2.2.3 八皇后问题 40
2.3 典型问题的 C++程序 43
2.4 小结 48
练习题 48

第 3 章 分治算法 50

3.1 分治算法的思想 51
3.2 排序问题中的分治算法 52
3.2.1 归并排序 53
3.2.2 快速排序 55
3.3 查找问题中的分治算法 57
3.3.1 折半查找 57
3.3.2 选择问题 59
3.4 组合问题中的分治算法 60
3.4.1 最大子段和问题 60
3.4.2 棋盘覆盖问题 62
3.5 典型问题的 C++程序 64
3.6 小结 70
练习题 71

第 4 章 贪心算法 72

4.1 贪心算法的思想 73
4.1.1 问题的提出 73
4.1.2 贪心算法设计思想 73
4.1.3 贪心算法的基本要素 74
4.1.4 贪心算法的求解过程 74
4.2 组合问题中的贪心算法 75
4.2.1 背包问题 75
4.2.2 多机调度问题 77

4.3 图问题中的贪心算法 78
4.3.1 单源最短路径问题 78
4.3.2 最小代价生成树 80
4.4 典型问题的 C++程序 84
4.5 小结 92
练习题 92
第 5 章 动态规划算法 94
5.1 动态规划算法的思想 95
5.2 查找问题中的动态规划算法 97
5.2.1 最优二叉搜索树 97
5.2.2 近似串匹配问题 100
5.3 图问题中的动态规划算法 102
5.3.1 多段图问题 102
5.3.2 每对结点间的最短距离 105
5.4 组合问题中的动态规划算法 108
5.4.1 0/1 背包问题 108
5.4.2 最长公共子序列 112
5.4.3 流水作业调度 115
5.5 典型问题的 C++程序 120
5.6 小结 125
练习题 126
第 6 章 回溯算法 128
6.1 回溯算法的思想 129
6.1.1 基本概念 129
6.1.2 基本思路 130
6.1.3 回溯算法的适用条件 132
6.1.4 回溯算法的效率估计 132
6.2 组合问题中的回溯算法 133
6.2.1 装载问题 133
6.2.2 0/1 背包问题 134
6.2.3 n 皇后问题 136
6.2.4 图的 m 着色问题 139
6.2.5 子集和数问题 141
6.3 图问题中的回溯算法 143
6.3.1 深度优先搜索 143
6.3.2 货郎（TSP）问题 143
6.3.3 最大团（MCP）问题 145
6.3.4 哈密顿环问题 146
6.4 算法效率的影响因素及改进途径 148
6.4.1 影响算法效率的因素 148
6.4.2 回溯算法的改进途径 148
6.5 典型问题的 C++程序 148
6.6 小结 165
练习题 165
第 7 章 分支限界算法 167
7.1 分支限界算法的思想 168
7.2 求最优解的分支限界算法 170
7.2.1 FIFO 分支限界算法 171
7.2.2 LC 分支限界算法 172
7.3 组合问题中的分支限界算法 173
7.3.1 0/1 背包问题 173
7.3.2 带限期的作业排序 175
7.4 图问题中的分支限界算法 179
7.4.1 旅行商问题 179
7.4.2 单源点最短路径问题 182
7.5 典型问题的 C++程序 184
7.6 小结 188
练习题 188
附录 实验指导 190
实验一 递归与分治算法 191
1.1 实验目的与要求 191
1.2 实验课时 191
1.3 实验原理 191
1.4 实验题目 191
1.5 思考题 192
实验二 贪心算法 192
2.1 实验目的与要求 192
2.2 实验课时 192
2.3 实验原理 192
2.4 实验题目 193
2.5 思考题 194
实验三 动态规划算法 194

3.1 实验目的与要求 194
3.2 实验课时 195
3.3 实验原理 195
3.4 实验题目 195
3.5 思考题 197
实验四 回溯算法 197
4.1 实验目的与要求 197
4.2 实验课时 197
4.3 实验原理 197
4.4 实验题目 198
4.5 思考题 199
实验五 分支限界算法 199
5.1 实验目的与要求 199
5.2 实验课时 200
5.3 实验原理 200
5.4 实验题目 200
5.5 思考题 203
参考文献 204

01 第1章 算法设计与分析基础

计算机系统中的任何软件都是按一个个特定的算法来予以实现的。用什么方法来设计算法，如何判定一个算法的性能，设计的算法需要多少运行时间、多少存储空间，这些问题都是在开发一个软件时必须考虑的。算法性能的好坏，直接决定了软件性能的优劣。

本章将主要介绍算法、算法设计、算法分析的基本概念，以及常见重要问题的类型和常用算法的设计方法。

1.1 算法概述

1.1.1 什么是算法

算法研究是计算机科学的主要任务之一。利用计算机解决一个实际问题时，首先是选择一个合适的数学模型表示问题，以便抽象出问题的本质特征，其次就是寻找一种算法，作为问题的一种解法。那么什么是算法，算法有什么基本特征，算法的组成有哪些？

1. 算法的定义

算法是解题方案的准确而完整的描述，也就是解题的方法和步骤。下面举两个例子来说明算法。

例 1.1 有 10 道数学应用题的作业，必须一道题、一道题地解答，解答每道题的过程是相同的：看题→思考→解答，然后验算检查有无错误，若没错就做下一道题；若有错，就重做。直到 10 道题都解答完，这次作业才算完成。解题的方法和步骤如图 1-1 所示。

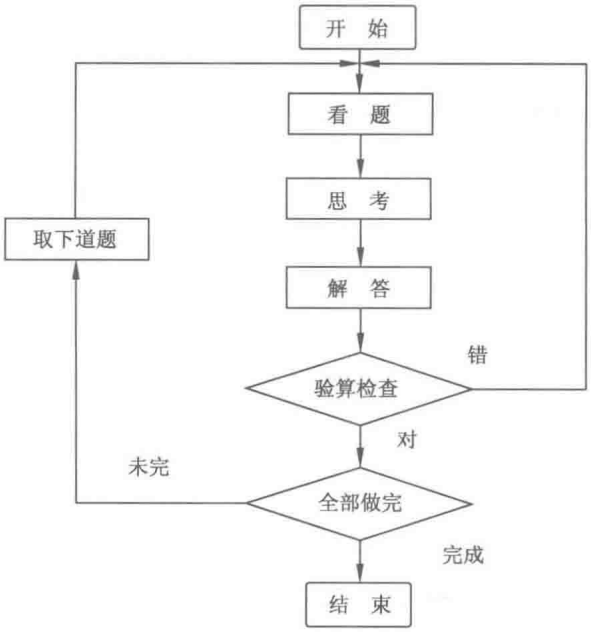

图 1-1 解题的方法和步骤

一个学生做任何作业都可以按这个“算法”执行，每次执行都产生相应的结果，这个算法的执行者是人而不是机器。

例 1.2 求任意两个整数 m 和 n（$0<m<n$）的最大公约数，称为欧几里德算法，记为 $\gcd(m, n)$。

作为例子，这里用了三种方法来解决这一问题，用以阐明算法概念的以下几个要点。

（1）算法的每一个步骤都必须清晰、明确。

（2）算法所处理的输入的值域必须仔细定义。

（3）同样一种算法可以用几种不同的形式来描述。

（4）可能存在几种解决相同问题的算法。

（5）针对同一个问题的算法可能会基于完全不同的解题思路，而且解题速度也会有显著不同。

【程序 1-1】欧几里德递归算法。

```
void swap(int &a, int &b)
    { int c;
      c=a; a=b; b=c;
    }
int rgcd(int m, int n)
    { if(m==0) return n;
      return rgcd(n%m, m);
    }
int gcd(int m, int n)
    { if (m>n) swap(m, n);
      return rgcd(m, n);
    }
```

【程序 1-2】欧几里德迭代算法。

```
int gcd(int m, int n)
    { if (m==0) return n;
      if (n==0) return m;
      if (m>n) swap(m, n);
      while(m>0)
         { int c=n%m; n=m; m=c; }
      return n;
    }
```

【程序 1-3】gcd 的连续整数检测算法。

```
int gcd(int m, int n)
    { int t ;
      if (m==0) return n;
      if (n==0) return m;
      int t=m>n?n:m;
      while (m%t || n%t) t--;
      return t;
    }
```

计算机科学中讨论的算法是由计算机来执行的，也可由人模拟它用笔和纸执行。算法中最低层的操作是对用存储器实现的变量进行赋值，这样整个算法就是一个信息变换器，对任意一组给定的输入值，产生一组唯一确定的输出结果值。

对算法（Algorithm）一词给出精确的定义是很难的。算法是用计算机解决某一类特定问题的一组规则的有穷集合，或者说是对特定问题求解步骤的一种描述，它是指令的有限序列。

也可以说，算法是将输入转化为输出的一系列计算步骤，它取某些数值或数值的集合作为输入，并产生某些值或值的集合作为输出。因此，算法是将输入转化为输出的一系列计算步骤。计算机科学中，算法已经逐渐成了用计算机解决问题的精确、有效方法的代名词。

简而言之，算法就是有效求出问题的解，对问题的求解过程进行精确的描述。那么一个算法应该具有哪些基本特征呢？

2. 算法的特征

算法通常具有以下几个特征。

（1）输入（Input）

一个算法可以有零个或多个输入。这些输入是在算法开始之前给出的量，它们取自于特定对象的集合，通常体现为算法中的一组变量。如【程序 1-1】中有两个输入 m、n。当然，有些算法也可

以没有输入，如求 10 以内素数的算法。

（2）输出（Output）

一个算法必须具有一个或多个输出，以反映算法对输入数据加工后的结果。这些输出是同输入有某种特定关系的量，实际上是输入的某种函数。不同取值的输入，产生不同的输出结果。没有输出的算法是没有意义的。如【程序 1-1】中的输出是输入 m、n 的最大公约数。

（3）确定性（Definiteness）

确定性指算法中的每一个步骤都必须是有明确定义的，必须是足够清楚的、无二义性的，不允许有模棱两可的解释，确定性保证了以同样的输入多次执行一个算法时，必定产生相同的结果，否则一定是执行者出了差错。例如，$b=2a$，多次输入 $a=2$，b 一定为 4。如【程序 1-1】中的两个输入 m、n 一定是从正整数集合中抽取的。

（4）可行性（Effectiveness）

算法中所有的操作都必须足够基本，使算法的执行者或阅读者明确其含义以及如何执行。它们可以通过已经实现的基本运算执行有限次数来实现；每种运算至少在原理上均能由人用纸和笔在有限的时间内完成。如“增加变量 x 的值”或“把 x 和 y 的最大公因子赋给 z”都不够明确，前者不知增加多少，后者不知如何去操作。

（5）有穷性（Finiteness）

算法的有穷性是指算法必须总能在执行有限步骤之后终止，且每一步的时间也是有限的。如果一个算法需要执行千万年，显然就失去了实用价值。如【程序 1-1】对输入的任意正整数 m、n，再把 m 除以 n 的余数赋值给 n，从而使 n 值变小，如此重复进行，最终使 $n=0$，算法终止。再如 1 除以 3 等于 0.3333…，如果规定保留小数点第几位后，按照四舍五入的方法计算，就可以确定一个值，而不是无穷计算下去。

3. 算法的基本要素

一个算法通常由两种基本要素组成：一是对数据对象的运算和操作，二是算法的控制结构。

（1）对数据对象的运算和操作

在一般的计算机系统中，基本的运算和操作有如下四类。

① 算术运算：主要包括加、减、乘、除等运算。

② 逻辑运算：主要包括与、或、非等运算。

③ 关系运算：主要包括大于、小于、等于、不等于等运算。

④ 数据传输：主要包括赋值、输入、输出等操作。

（2）算法的控制结构

一个算法的功能不仅取决于选用的操作，而且还与各操作之间的执行顺序有关，算法中各操作之间的执行顺序称为算法的控制结构。

一个算法一般都可以用顺序、选择、循环（当型循环或直到型循环）三种控制结构组成。如例 1.1 中，“看题→思考→解答”是顺序执行，“验算检查”对错是选择控制结构，对就继续执行，错则重做是循环控制结构。

4. 算法的描述工具

描述算法可以有多种方式。

（1）自然语言

自然语言就是人们日常使用的语言，可以是汉语、英语或其他语言。自然语言描述方法就是直接将设计者完成任务的思维过程用其母语描述下来。用自然语言描述算法非常接近人类的思维习惯，是一种非形式描述方法。初学者可以首先使用这种方法描述完成任务的步骤，然后再转换成其他描述。

（2）流程图

流程图是用规定的图形、流程线、文字说明表示算法的方法，是一种图形方式的描述手段。流程图可以清晰地描述出完成解题任务的方法及步骤，它是使用最早的算法描述工具。它的优点是非常直观。

（3）N-S 流程图

N-S 流程图是 1973 年提出的一种新的流程图形式，其名称来源于提出它的两位英国学者 I.Nassi 和 B.Shneiderman。N-S 流程图也是图形方式的描述手段，N-S 流程图简称 N-S 图。

在 N-S 图中去掉了容易引起麻烦的流程线，不允许随意使用转移控制，全部算法都写在一个框内，框内还可以包含其他框。这种流程图适用于结构化程序设计，能清楚地显示出程序的结构，是一种结构化的流程图。

N-S 图的优点是简洁，但当嵌套层数太多时，内层的方框将越画越小，从而会影响图形的清晰度。

（4）伪代码

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法，是一种描述语言。它不用图形符号，因此比较紧凑、简洁，也比较好懂，与程序语言的形式非常接近，也容易转化为高级语言程序，是常用的算法描述方法。

伪代码只是一种描述程序执行过程的工具，是一种在程序设计过程中表达想法的非正式的符号系统，是面向读者的，不能直接用于计算机，实际使用时还需转换成某种计算机语言来表示。

无论采用何种算法描述方式，若最终要在计算机中执行，都需要转换为相应的计算机语言程序。

5. 算法与程序和数据结构的关系

（1）算法与程序

算法的概念与程序（Program）十分相似，但也有很大的不同。

算法代表了对特定问题的求解，算法是行为的说明，是一组逻辑步骤。而计算机程序则是算法用某种程序设计语言的表述，是算法在计算机上的具体实现。执行一个程序就是执行一个用计算机语言表述的算法。因此算法也常常被称为一个可行的过程。

算法在描述上一般使用半形式化的语言，而程序是用形式化的计算机语言描述的，是使用一些特殊编程语言表达的算法。算法对问题求解过程的描述可以比用程序描述粗略些，算法经过细化以后可以得到计算机程序。

一个算法可以用不同的编程语言编写出不同的程序，但它们遵循的逻辑步骤是相同的，它们都表达同样的算法，它们不是同样的程序。例如，对于同一个菜肴的制作步骤，可以分别使用英语、法语和日语写成。这是不同的三个菜谱，但是它们都表达同一个操作步骤。

程序并不都满足算法所要求的特征，例如，“操作系统”是一个在无限循环中执行的程序，不具备“有穷性”的特征，因而“操作系统”是一种程序而不是一个算法。

（2）算法与数据结构

不了解施加于数据上的算法就无法决定如何构造数据，可以说算法是数据结构的灵魂；相反地，算法的结构和选择又常常在很大程度上依赖于数据结构，数据结构则是算法的基础。算法与数据结构是密不可分的，二者缺一不可。因此有人说："算法＋数据结构＝程序"。

1.1.2 学习算法的重要性

算法是计算机科学的基础，更是程序的基石，只有具有良好的算法基础才能成为训练有素的软件人才。对计算机类专业的学生来说，学习算法是十分必要的。因为你必须知道来自不同计算领域的重要算法，你也必须学会设计新的算法、确认其正确性并分析其效率。

随着计算机应用的日益普及，各个应用领域的研究人员和技术人员都在使用计算机求解他们各自专业领域的问题，他们需要设计算法、编写程序、开发应用软件，所以学习算法对于越来越多的人来说变得十分必要。

计算机的操作系统、语言编译系统、数据库管理系统以及各种各样的计算机应用软件，都要用具体的算法来实现。因此算法设计与分析是计算机科学与技术的一个核心问题，也是一门重要的专业基础课程。

通过对算法设计与分析这门课程的学习，读者就能掌握算法设计与分析的方法，再利用这些方法去解决软件开发中所遇到的各种问题，去设计相应的算法，并对所设计的算法做出科学的评价。无论是计算机专业技术人员，还是使用计算机的其他专业技术人员，算法设计与分析都是非常重要的。

1.2 问题的求解过程

软件开发的过程就是计算机求解问题的过程，使用计算机解题的核心就是进行算法设计，算法是为解决特定问题而采取的有限操作步骤，是对解决方案准确而完整的描述。

算法是精确定义的，可以认为算法是问题程序化的解决方案。

1.2.1 问题及问题的求解过程

当前情况和预期的目标不同就会产生问题，求解问题（Problem Solving）是寻找一种方法来实现目标。问题的求解过程（Problem Solving Process）是人们通过使用问题领域的知识来理解和定义问题，并凭借自身的经验和知识去选择和使用适当的问题求解策略、技术和工具，将一个问题的描述转换成对问题求解的过程，如图 1-2 所示。

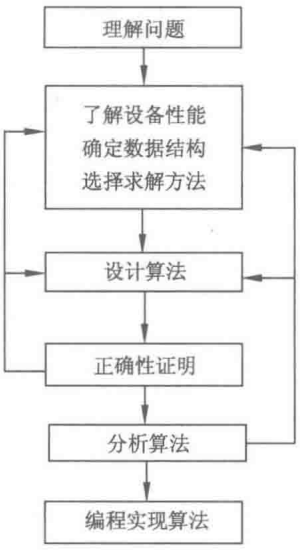

图 1-2 问题求解的过程

计算机求解问题的关键之一是寻找一种问题求解策略（Problem Solving Strategy），得到求解问题的算法，从而得到问题的解。

一个计算机程序的开发过程就是使用计算机求解问题的过程。软件工程（Software Engineering）将软件开发和维护过程分成若干阶段，称为系统生命周期（System Life Cycle）或软件生命周期。

通常把软件生命周期划分为：分析（Analysis）、设计（Design）、编

码（Coding or Programming）、测试（Testing）和维护（Maintenance）5 个阶段。前 4 个阶段属于开发期，最后一个阶段处于运行期。

算法设计的整个过程，可以包含问题需求的说明、数学模型的拟制、算法的详细设计、算法的正确性验证、算法的实现、算法分析、程序测试和文档资料的编制。在此我们只关心算法的设计和分析。

现在给出在算法设计和分析过程中所要经历的一系列典型步骤。

（1）理解问题（Understand the Problem）：在设计算法前首先要做的就是完全理解所给出的问题。明确定义所要求解的问题，并用适当的方式表示问题。

（2）设计方案（Devise a Plan）：求解问题时，考虑从何处着手，考虑选择何种问题求解策略和技术进行求解，以得到问题求解的算法。

（3）实现方案（Carry Out the Plan）：实现求解问题的算法，使用问题实例进行测试、验证。

（4）回顾复查（Look Back）：检查该求解方法是否确实求解了问题或达到了目的。

（5）评估算法，考虑该解法是否可以简化、改进和推广。

1.2.2　算法设计与算法表示

1. 算法问题的求解过程

算法问题的求解过程本质上与一般问题的求解过程是一致的。求解一个算法问题，需要先理解问题。通过最小阅读对问题的描述，充分理解所求解的问题。

算法一般分为两类：精确算法和启发式算法。精确算法（Exact Algorithm）总能保证求得问题的解；启发式算法（Heuristic Algorithm）通过使用某种规则、简化或智能猜测来减少问题求解的时间。

对于最优化问题，一个算法如果致力于寻找近似解而不是最优解，被称为近似算法（Approximation Algorithm）。如果在算法中需做出某些随机选择，则称为随机算法（Randomized Algorithm）。

2. 算法设计策略

使用计算机的问题求解策略主要指算法设计策略（Algorithm Design Strategy）（技术）。算法设计策略是使用算法解题的一般性方法，可用于解决不同计算领域的多种问题。这是创造性的活动，学习已经被实践证明是有用的一些基本设计策略是非常有用的。值得注意的是要学习设计高效的算法。算法设计方法主要有：分治策略、贪心算法、动态规划、回溯法、分支限界法等。我们将在后面的章节中陆续介绍。

3. 算法的表示

算法需要用一种语言来描述，算法的表示是算法思想的表示形式。显然，用自然语言描述算法时，往往一个人认为明确的操作，另一个人却觉得不明确或者尽管两个人都觉得明确了，但实际上有着不同的理解，因此，算法应该用无歧义的算法描述语言来描述。

计算机语言既能描述算法，又能实际执行。在这里，我们将采用 C++语言来描述算法。C++语言的优点是数据类型丰富，语句精炼，功能强，效率高，可移植性好，既能面向对象又能面向过程。用 C++语言来描述算法可使整个算法结构紧凑，可读性强，便于修改。

在课程中，有时为了更好地阐明算法的思路，我们还采用 C++语言与自然语言相结合的方式来描述算法。

1.2.3 算法确认和算法分析

确认一个算法是否正确的活动称为算法确认（Algorithm Validation），其目的在于确认一个算法是否能正确无误地工作，即证明算法对所有可能的合法输入都能得出正确的答案。

1. 算法证明

算法证明与算法描述语言无关。使用数学工具证明算法的正确性，称为算法证明。有些算法证明简单，有些算法证明困难。在本课程中，仅对算法的正确性进行一般的非形式化的讨论和通过对算法的程序实现进行测试。

证明算法正确性的常用方法是数学归纳法。如【程序 1-1】中求最大公约数的递归算法 rgcd，可用数学归纳法证明如下：

设 m 和 n 是整数，$0 \leqslant m < n$。若 $m=0$，则因 gcd(0, n)=n，程序 rgcd 在 $m=0$ 时返回 n 是正确的。归纳法假定当 $0 \leqslant m < n < k$ 时，函数 rgcd(m, n)能在有限时间内正确返回 m 和 n 的最大公约数，那么当 $0 < m < n = k$ 时，考察函数 rgcd(m, n)，它将具有 rgcd($n\%m$, m)的值。这是因为 $0 \leqslant n\%m < m$ 且 gcd(m, n)=gcd(n % m, m)，故该值正是 m 和 n 的最大公约数，证毕。

如果要表明算法是不正确的，举一个反例，即给出一个能够导致算法不能正确处理的输入实例就可以。

2. 算法测试

程序测试（Program Testing）是指对程序模块或程序总体，输入事先准备好的样本数据（称为测试用例，Test Case），检查该程序的输出，来发现程序存在的错误及判定程序是否满足其设计要求的一项活动。

调试只能指出有错误，而不能指出它们不存在错误。测试的目的是发现错误，调试是诊断和纠正错误。大多数情况下，算法的正确性验证是通过程序测试和调试排错来进行的。

3. 算法分析

根据算法分析与设计的步骤，在完成算法正确性检验之后，要做的工作就是算法分析。

算法分析（Algorithm Analysis）是对算法利用时间资源和空间资源的效率进行研究。算法分析活动将对算法的执行时间和所需的存储空间进行估算。算法分析不仅可以预计算法能否有效地完成任务，而且可以知道在最好、最坏和平均情况下的运算时间，对解决同一问题不同算法的优劣做出比较。

当然在算法写成程序后，便可使用样本数据，实际测量一个程序所消耗的时间和空间，这称为程序的性能测量（Performance Measurement）。

1.3 算法的复杂性分析

一个问题如果采用了合适的算法，其解决问题的速度将会大大提高。那么评价一个算法的好坏，主要看执行算法时需要花费的计算机 CPU 时间的多少和需要占用的计算机存储空间的大小。因此，

算法的分析就是对其时间效率和空间效率两个方面进行比较。

这里讨论衡量算法效率的时间复杂度和空间复杂度。

1.3.1　算法评价的基本原则

算法的优劣是经过分析后得出的结果，而为判断算法的效率对其进行的分析即算法分析。但是效率分析并不是算法分析的唯一目的，虽然算法追求的目标是速度，但算法必须首先正确才有存在的意义。

不过，正确性和时间分析并不是算法分析的唯一任务，如果两个算法的时间效率是一样的，就要对算法实现的空间进行比较，空间使用较少的为优。在某些情况下，两个算法的时间、空间效率都有可能相同或相似，这时就要分析算法的其他属性，如稳定性、健壮性、实现难度等，并以此来判断到底应该选择哪一个算法。通常一个好的算法应该考虑达到以下目标。

1. 正确性

一个好的算法的前提就是算法的正确性（Correctness）。不正确的算法没有任何意义。

在给定有效输入后，算法经过有限时间的计算，执行结果满足预先规定的功能和性能要求，答案正确，就称算法是正确的。算法应当满足具体问题的需求，否则，算法的正确与否的衡量准则就不存在了。

“正确”一词的含义在通常用法上有很大差别，大体可分为四层。

（1）程序不含语法错误。

（2）程序对几组输入数据能够得出满足规格说明要求的结果。

（3）程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果。

（4）程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

达到第 4 层意义上的正确是极为困难的，所有不同输入数据的数据量大得惊人，逐一验证是不现实的，在实际上，通常以第 3 层意义下的正确作为一个程序是否合格的标准。

对于大型程序，可以将它分解为小的相互独立的模块，分别进行验证。而小模块程序则可以使用数学归纳法、软件形式方法等加以验证。

2. 可读性（Readability）

算法主要是为了方便用户的阅读与交流，其次才是机器的执行。因此，算法应该易于理解、调试和修改，可读性好则有助于用户对算法的理解。

3. 健壮性和可靠性

健壮性（Robustness）是指当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫名其妙的输出结果。即当程序遇到意外时，能按某种预定方式做出适当处理。例如，求一个凸多边形面积的算法，是采用求各三角形面积之和的策略来解决问题的。当输入的坐标集合表示的是一个凹多边形时，不应继续计算，而应报告输入错误，并且返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

正确性和健壮性是相互补充的。

程序的可靠性指一个程序在正常情况下能正确地工作，而在异常情况下也能做出适当处理。

4. 效率

效率（Efficiency）包括运行程序所花费的时间以及运行这个程序所占用的存储空间。算法应该有效使用存储空间，并具有高的时间效率。

通俗地说，效率是指算法的执行时间，对于同一个问题，如果有多个算法可以解决，执行时间短的效率高，存储量需求指算法执行过程中所需要的最大存储空间，效率与低存储量需求这两者都与问题的规模有关，求 100 个人的平均分与求 10000 个人的平均分所花的执行时间或运行空间显然有一定差别。

对于规模较大的程序，算法的效率问题是算法设计必须面对的一个关键问题，目标是设计复杂性尽可能低的算法。

5. 简明性

简明性是指算法应该思路清晰、层次分明、容易理解、利于编码和调试，即算法简单、程序结构简单。

简单的算法效率不一定高，要在保证一定效率的前提下力求得到简单的算法。简明性（Simplicity）是算法设计人员努力争取的一个重要特性。

6. 最优性（Optimality）

最优性指求解某类问题中效率最高的算法，即算法的执行时间已达到求解该类问题所需时间的下界。最优性与所求问题自身的复杂程度有关。

例 1.3 在 n 个不同的数中找最大数的算法 Find max(L, n)。

输入：数组 L，项数 n。

输出：L 中的最大项 max。

```
max=L[1];  i=2;
while( i<=n )
{   if (max<L[i] )  max=L[i];
    i=i+1;
}
```

因为 max 是唯一的，其他的 n–1 个数必须在比较后被淘汰。一次比较至多可淘汰 1 个数，所以至少需要 n–1 次比较。即在有 n 个数的数组中找数值最大的数，并以比较作为基本运算的算法至少要做 n–1 次比较。Find max 算法是最优算法。

一般来说，正确性和可读性都比效率重要，一个在某些情况下会得出错误结果的算法，即使效率再高，也是没有意义的。当然在基本保证正确的前提下，效率也是非常重要的。

1.3.2 影响程序运行时间的因素

一个程序的运行时间是程序运行从开始到结束所需的时间。影响程序运行时间的因素主要有以下几方面。

1. 程序所依赖的算法

求解同一个问题的不同算法，其程序运行时间一般不同。一个好的算法运行时间较少。算法自身的好坏对运行时间的影响是根本的和起作用的。