第 1 章

Python算法入门

本章概述

要想较快地解决实际问题，可以通过算法来实现。与算法息息相关的是数据结构，算法和数据结构都是学习编程的基础。本书将通过 Python 语言探索算法的奥秘。算法虽然不依赖于任何编程语言，但要想学习算法，必须通过编程语言来实现。

知识导读

本章要点（已掌握的在方框中打钩）

- ☐ 算法的特性
- ☐ 程序、编程与算法之间的关系
- ☐ 算法的表示方法
- ☐ 基本数据结构

1.1 认识算法

在计算机程序中，可以通过某个已有的方法来解决某个问题。这种通过计算机程序实现的解决问题的方法称为算法。

1.1.1 什么是算法

算法是一种有序且明确的指令集，主要用于解决特定的运算或逻辑问题。算法会告诉计算机如何处理数据，以及如何执行特定的任务。

算法是计算机编程的重要组成部分，它规定了计算机执行特定任务的具体步骤和方法。算法既可以非常简单，也可以非常复杂，这取决于它们旨在解决的问题。不同的算法可能会使用不同的时间、空间或效率来完成同一个任务。算法的好坏可以用空间复杂度和时间复杂

度来衡量。

在计算机科学领域，算法可以接受输入，按照一系列计算步骤对其进行处理，并产生输出或结果。

（1）输入 / 输出：算法通常有一个或多个输入，并且产生一个输出。

（2）确定性：算法中的每条指令都应该清晰、明确地描述如何执行计算。每次给定相同的输入，它都会产生相同的输出。

（3）有穷性：对于任何输入，算法必须在有限的步骤内结束。

（4）有效性：算法在有限的时间内为所有可能的有效输入一致且准确地生成有意义和正确结果的能力。

常见的算法类型如下：

（1）排序算法：如冒泡排序、选择排序、插入排序、快速排序、归并排序等。

（2）搜索算法：如线性搜索、二分搜索等。

（3）图论算法：如深度优先搜索、广度优先搜索、最小生成树算法、最短路径算法等。

（4）字符串处理算法：如字符串匹配的 KMP 算法、回文检测等。

（5）动态规划算法：如背包问题等。

（6）贪心算法：如活动选择问题、最小生成树问题等。

（7）分治算法：如快速排序、归并排序等。

1.1.2 算法的特性

一个好的算法是解决问题的正确程序。好的算法应该能够处理大型数据集，并能够适应不断变化的环境或要求；好的算法不仅是易于理解和维护的，而且还应具有清晰的文档和组织良好的代码，以便日后进行修改和调试。

一个好的算法不仅仅是解决一个问题，而是提供一个强大且适应性强的框架，用于解决各种环境中的各种问题。

算法通过精心设计一系列的步骤，不仅为该问题提供了有效的解决方案，而且还考虑了可扩展性、效率和灵活性。

因此，好的算法应该具有以下几个特征：

（1）有限性。算法的有限性是指其执行的步骤是有限的。也就是说在有限的时间内，算法能够终止并计算出该问题的结果。有限性可以保证算法的高效性和实用性，避免该算法陷入死循环或得不到正确的结果。例如，在排序算法中，无论输入数据的数量有多庞大，算法总能在有限的步骤内对该数据进行排序并输出结果。

（2）确定性。算法的确定性是指该算法中的每一个步骤都是非常明确的，并且可以按照设定的方法来执行相应的步骤。在相同的输入条件下，算法无论执行多少次都会产生相同的结果。确定性既保证了算法的可预测性，又保证了算法的可靠性，从而使算法成为解决复杂问题的有力工具。

（3）可输入性。输入可以作为算法的初始条件，算法的输入可以是多个或 0 个，会影响算法

的执行过程和结果。算法通过可输入性能够解决不同的问题，因此使得算法更加灵活和通用。

（4）可输出性。算法根据输入的内容，执行完相应的步骤后，必然会产生一个或多个输出结果。输出结果是对输入问题的解决处理，反映出算法的执行效果。如果一个算法没有输出结果，那么该算法是没有意义的，因为无法显示出该算法的价值。

（5）可行性。算法中的每个步骤都必须是在现实条件下可以实现的。可行性要求该算法中的运算和操作必须是准确的、可行的，并且能够在有限的时间内可以完成。可行性既保证了算法的实际应用价值，又使得算法能够在解决实际问题的过程中得到广泛应用。

1.2　程序、编程与算法之间的关系

算法等于流程控制和数据结构的总和。算法可以说是程序的内容，而编程则是实现算法的过程。学习编程的主要目的是通过实现算法来解决实际问题。

1.2.1　算法与程序

算法与程序都属于编程的核心概念，它们不仅有着紧密的联系，而且还有所区别。算法与程序之间的主要区别体现在以下几个方面：

1. 定义方面的不同

算法是解决问题的明确指令，该指令通过规范一定的输入，可以在有限的时间内获取特定的输出。算法是对具体问题求解步骤的描述，它不包括具体的编程语言实现。算法比较注重解决问题的逻辑和方法，强调的是“怎么做”。

程序是一系列指令的有序集合，是实现算法的具体手段，主要用于告诉计算机如何执行特定的任务。由于程序是通过使用特定的程序设计语言实现的，因此当计算机在执行特定的任务时，可以直接理解并执行这些指令。程序比较注重将算法转化为计算机可以执行的代码，强调的是“怎么做出来”。

总的来说，算法强调的是解决问题的思路和方法，而程序强调的是解决问题的思路和方法在计算机上是如何实现的。算法是程序的灵魂，程序是算法的主体。如果没有算法，程序就像是在大海上航行迷失了方向；如果没有程序，算法就无法在计算机上执行。

2. 特性方面的不同

算法的特性如下：

（1）有限性：算法在执行有限的步骤后结束。

（2）确定性：算法中的每一步都有明确的定义，不会产生歧义。

（3）可行性：算法中的所有操作都可以通过已经实现的有限的基本操作运算来实现。

（4）输入：算法可以有 0 个或多个输入。

（5）输出：算法必然有一个或多个输出，以表示算法的结果。

程序的特性如下：

（1）有序性：为了便于计算机的执行，程序中的指令都是按照特定的顺序排列的。

（2）明确性：为了便于计算机的理解及执行，程序的指令都是明确的、简洁的。

（3）逻辑性：程序需要合理的逻辑结构，以确保计算机能够按照预设的方案完成任务。

3. 执行方面的不同

算法并不需要使用某种特定的语言来实现的，它更侧重于人对算法方法的理解。

而程序则是在某种特定的程序设计语言的基础上实现的，以便于计算机的理解和执行。

4. 执行时间方面的区别

算法所描述的步骤都是有限的，它会在执行有限的步骤之后结束。

而对于程序来说，当没有遇到终止条件或发生错误时，程序则可以无限地执行下去。

1.2.2 算法与编程

算法是解决问题的方法或步骤，而编程则是将算法通过某种特定的语言实现的过程。算法是编程的基础，它为编程提供了解决问题的思路和方法。

算法主要侧重于问题的描述和解决方案的设计。例如，常见的排序算法包括冒泡排序、快速排序、插入排序等，这些排序算法描述了如何按照某种规则对输入的数据进行排序，而不涉及具体编程语言的实现过程。

编程则是通过某种特定的编程语言将算法转化为计算机可以理解并执行的代码的过程。编程是实现算法的手段之一，它可以通过多种编程语言来实现算法。编程语言是用于描述计算机行为的形式语言，通过编程语言编写的代码可以在计算机上运行和执行特定的任务。

编程需要考虑的因素有很多，如数据的存储、指令的选择、代码的优化等多个方面。通过编程，可以将算法应用于解决各种复杂的实际问题。

1. 算法与编程的关系

算法和编程是密不可分、相互依存的。算法为编程提供了解决问题的思路和方法，而编程则将算法转化为计算机可执行的代码。

在解决实际问题的应用中，开发人员会根据问题的性质选择合适的算法，然后通过某种编程语言来实现该算法，最终形成可以在计算机上运行的程序。优秀的算法设计可以提高编程的效率和质量，使程序更加高效和可维护。同时，掌握编程技能也有助于更好地理解和应用各种算法。

总的来说，算法与编程是计算机中不可或缺的两个重要部分。它们共同构成了解决问题的框架和实现方法，是开发高效、可靠软件的基础。

2. 算法在编程中的应用

算法在编程中扮演着至关重要的角色。

（1）排序算法：包括快速排序、插入排序、归并排序和堆排序等。由于排序算法能够高效地对大量数据进行排序，因此广泛应用于对数据的处理。

（2）图论算法：包括深度优先搜索（DFS）和广度优先搜索（BFS）等。图论算法能够

帮助分析和处理用户关系数据，优化网络性能和用户体验。

（3）搜索算法：包括线性搜索、二分搜索等。搜索算法常用于在数据库或其他数据结构中查找特定信息。例如，在搜索引擎中，搜索算法会根据关键词在网页上进行查找，并根据查找到的内容进行排序。

（4）机器学习算法：在机器学习领域，包括决策树、支持向量机和神经网络等算法。机器学习算法可以帮助构建高效的分类和预测模型，提高模型的准确性和效率。

1.3　算法的表示方法

算法是解决问题的方法和步骤，它描述了解决特定问题或执行特定任务的明确指令序列。算法的表述方式有多种，常见的有自然语言、流程图、N-S 流程图和计算机语言等。

1.3.1　用流程图表示算法

通过流程图表示算法是最常见的一种方式。流程图可以清晰地表示算法的 3 种基本控制结构：顺序结构、选择结构和循环结构。通过流程图，可以提高算法的清晰度和可读性，使复杂的算法更加易懂。

（1）顺序结构：流程图中的顺序结构用矩形框表示处理过程，用箭头指向表示流程的方向。

（2）选择结构：选择结构用菱形框表示判断，判断必须有两个分支（满足条件或者不满足条件），箭头从菱形框分别指向两个分支。

（3）循环结构：循环结构在流程图中通过特定的符号和箭头指向表示循环的开始和结束，以及循环体内的操作。

常见的流程图符号如图 1-1 所示。

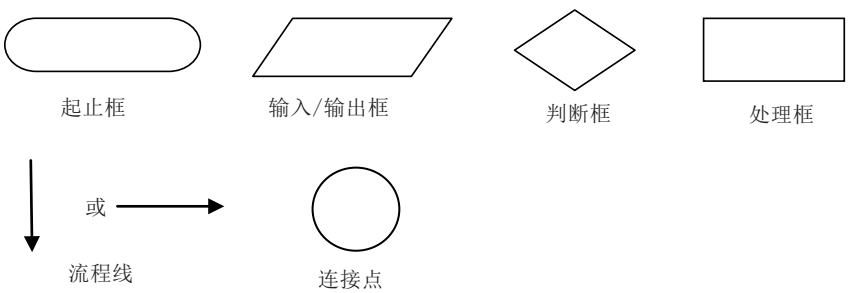

图 1-1　常见的流程图符号

起止框主要用来标识算法的开始和结束；判断框的作用是对一个给定的条件进行判断，并根据给定的条件是否成立来决定如何执行后面的操作；连接点是指将画在不同地方的流程

线连接起来。

使用流程图绘制算法的步骤如下：

（1）确定算法的输入和输出：明确算法需要哪些输入数据、最终会输出什么结果，以及算法的中间步骤。

（2）选择流程图符号：根据算法的步骤和结构，选择恰当的流程图符号（如矩形框、菱形框等）。

（3）绘制流程图：在绘图区域，按照算法的执行顺序，使用流程线将各个流程图符号连接起来，并用箭头指向表示流程的方向，以形成完整的流程图。

（4）标注关键信息：在每个流程图符号中标注关键信息，如输入 / 输出变量、判断条件、操作结果等。

（5）检查流程：检查流程图是否完整、准确，确保没有遗漏或错误的步骤。

例如，求两个整数 m 和 n 的最大公约数，流程图如图 1-2 所示。

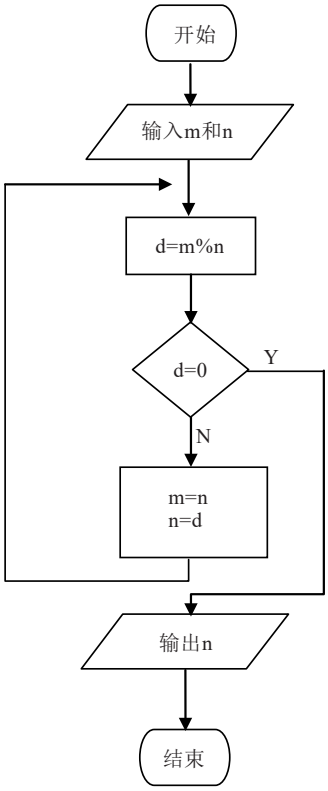

图 1-2　求两个整数 m 和 n 的最大公约数的算法流程图

1.3.2　用 N-S 流程图表示算法

N-S 流程图也被称为盒式图或 NS 图，是由美国人 I.Nassi 和 B.Shneiderman 共同提出

的，故以他们名字的首字母命名。

N-S 流程图是一种用于表示算法逻辑的图形化工具，它使用矩形框来表示基本的操作或步骤，并通过箭头来指示流程的方向。相比于传统的流程图，它去掉了原来所有的流程线，将全部的算法写在一个矩形框内，使其更加简洁、清晰，并且避免了流程线交叉的问题。

N-S 流程图同样也有 3 种基本结构。

（1）顺序结构的 N-S 流程图如图 1-3 所示。

（2）选择结构的 N-S 流程图如图 1-4 所示。

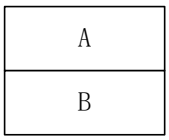

图 1-3　顺序结构

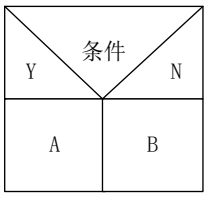

图 1-4　选择结构

例如，输入某个数，判断该数是否是奇数，此程序的选择结构的 N-S 流程图如图 1-5 所示。

（3）循环结构。

① while、for 循环：只有当满足一定条件时，才重复执行某操作；条件不满足时，停止执行。while、for 循环的循环结构如图 1-6 所示。

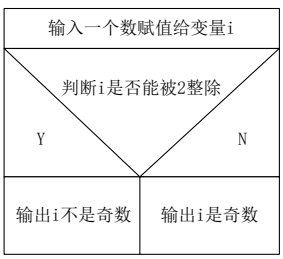

图 1-5　判断是否是奇数

图 1-6　while、for 循环的循环结构

例如，求 1 ～ 100（包括 1 和 100）所有的整数之和，该程序的 N-S 流程图如图 1-7 所示。

② do-while 循环：先执行一次操作，再判断条件是否成立，若条件满足，则继续执行，直到条件不满足时，停止执行。do-while 循环的循环结构如图 1-8 所示。

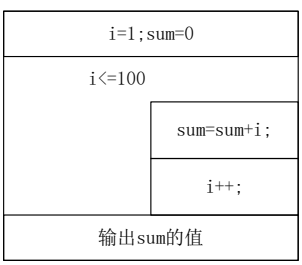

图 1-7　求 1 ～ 100（包括 1 和 100）所有的整数之和的 N-S 流程图 1

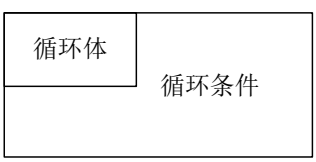

图 1-8　do-while 循环的循环结构

例如，求 1 ～ 100（包括 1 和 100）所有的整数之和，该程序的 N-S 流程图如图 1-9 所示。

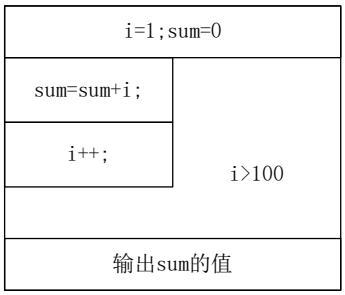

图 1-9　求 1 ～ 100（包括 1 和 100）所有的整数之和的 N-S 流程图 2

1.3.3　用计算机语言表示算法

由于计算机无法直接识别流程图，因此在使用流程图描述出一个算法后，还需要将它转换为计算机能够识别的语言程序。只有使用计算机语言编写的程序才能被计算机执行。

使用计算机语言表示算法通常包括以下几个步骤：

（1）选择编程语言。首先需要选择一种合适的编程语言来表示算法。常见的编程语言包括 Python、Java、C++、JavaScript、Go 等。每种语言都有其特定的语法和特性，但算法的基本概念及使用在不同的编程语言中都是相通的。

（2）定义问题。明确要解决的问题、输入及输出，这是算法设计的基础。

（3）设计算法。在进行编程之前，要先在逻辑上设计出算法，如画流程图、写伪代码或者描述算法的步骤等。

（4）编写代码。将设计好的算法使用编程语言转换为计算机能够理解的代码。

例如，使用 Python 语言表示计算两个数之和的算法，代码如下所示：

```
def add_numbers(a, b):
    # 计算两个数之和
    return a + b
# 例如：a=10，b=20
result = add_numbers(10, 20)
# 输出 30
print(result)
```

1.4　基本数据类型

数据类型是学习算法的基础，本节主要学习 Python 算法的基本数据类型，包括列表、

元组、字典和集合等。

无论是哪种编程语言，数据结构都用于存储和操作复杂的数据。在 Python 中，数据结构的基本数据类型如表 1-1 所示。

表 1-1　Python 的基本数据类型

数 据 类 型	说　　明
列表	有序的、可嵌套的、可变的元素序列
元组	有序的不可变元素序列
字典	键值对的无序集合
集合	元素的无序集合

1.4.1　列表

列表是元素按顺序排列构成的有序集合，主要用于存储一组有序的元素。每个元素在列表中都有一个固定的位置，即索引，可以通过索引访问特定元素。列表里的元素可以是整数、实数或字符串，甚至还可以是列表、元组、字典等，并且同一个列表中的元素类型也可以不同。列表中的元素是可以被修改的。

1. 列表的创建

可以使用 list() 函数直接创建列表。列表的创建使用一对方括号“[]”，并使用逗号“,”作为元素的分隔符。当“[]”内不存在任何元素时，便创建了一个空列表。例如：

```
list = []                  # 创建一个空列表
list = [5]                 # 创建一个只有一个元素的列表
list = [1, 2, 3]           # 创建一个有 3 个元素的列表
```

2. 访问列表中的元素

在 Python 中，访问列表中的元素就是输出列表中的内容。

（1）使用 print() 函数直接输出内容。

```
list=['hello','python',['我','很','好']]
print(list)
print(list[2])
```

输出：

```
['hello', 'python', ['我', '很', '好']]
['我', '很', '好']
```

从上述输出结果中可以看出，输出的方式有全输出和部分输出两种。全输出方式在输出时包括左右两侧的中括号；而部分输出方式是通过列表的索引获取指定的元素。并且在输出单个列表元素时，不包括中括号（最外围的）；如果输出字符串，则还不包括左右的引号。

（2）访问列表中特定的元素。

```
list=['a',1,2,'b',3.14]          # 创建一个包含不同数据元素的列表
print(list[4])                   # 访问列表 list 中的第 5 个元素（索引为 4）
```

输出：

```
3.14
```

列表元素的索引从 0 开始计数，即第一个元素的索引为 0。

3. 列表的操作函数

Python 中提供了很多便于对列表元素进行操作的函数。常见的列表操作函数如表 1-2 所示。

表 1-2　列表操作函数

列 表 函 数	说　　明
list.append(x)	在列表最后添加一个元素 x
list.extend(L)	在列表中添加另一个列表 L
list.insert(i,x)	在列表中的指定位置（i）插入数据（x）
list.pop(i)	将列表中第 i 个位置的元素取出并删除该元素
list.remove(x)	删除列表中的指定元素（有多个则只删除第一个），若指定元素不存在则报错
list.reverse()	将列表中的元素反转
list.sort()	将列表中的元素排序

（1）添加元素和列表。

使用 append() 函数可以在已有列表的基础上添加元素。例如：

```
list=['A','B','C']
print("此时列表中的内容 :", list)
list.append('D')
print("此时列表中的内容 :", list)
```

输出：

```
此时列表中的内容 : ['A', 'B', 'C']
此时列表中的内容 : ['A', 'B', 'C', 'D']
```

除了可以在已有列表中添加单个元素，还可以添加新的列表。例如：

```
list.append(['a','b','c'])
print("此时列表中的内容 :", list)
```

输出：

```
此时列表中的内容 : ['A', 'B', 'C', 'D', ['a', 'b', 'c']]
```

（2）插入元素。

可以在列表中的某个位置（按索引坐标排序）插入新的元素。例如：