第3章 计算机算法与数据结构

数据结构与算法是计算机程序设计的重要理论技术基础，是计算机学科的核心课程，在计算机技术发展过程中起着重要的推动作用。本章以基本的数据结构和算法的设计策略为知识单元，简要地介绍数据结构的知识、计算机算法的设计与分析方法，主要内容包括算法基础，线性表、树和图等常见数据结构以及简单的查找与排序算法等。

3.1 算法基础

对于计算机科学来说，算法(algorithm)的概念至关重要。通俗地讲，它是指解决问题的方法或过程。在软件项目开发中，一个好的算法是高质量程序设计的关键。本节以一个简单例子简要的介绍算法的基础知识。

3.1.1 算法的基本概念

算法代表着用系统的方法描述解决问题的策略机制。也就是说，对一定规范的输入，能够在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。如求解 1＋2＋3＋4＋…＋1000 的和 SUM。

算法一：首尾相加法。基本步骤如下。

第一阶段：取首尾各数分别相加。

第一步：取第一个数和最后一个数相加求得和 $S1$，即 $S1=1+1000=1001$。

第二步：取第二个数和倒数第二个数相加求得和 $S2$，即 $S2=2+1000=1001$。

⋮

第 500 步：取第 500 个数和倒数第 500 个数相加求得和 $S500$，即 $S500=500+501=1001$。

第二阶段：求和式 $S1+S2+\cdots+S500$。

由于 $S1$ 到 $S500$ 这 500 个数的值都为 1001，所以利用乘法求得 SUM＝1001＊500＝500500，得到最后结果 SUM＝500500。

算法二：利用等差数列。

由于待求解的和式为一等差数列且公差为 1，所以利用等差数列的求和公式很容易得到 SUM＝(1＋1000)＊1000/2＝1001＊500＝500500，同样得到结果。

基本步骤如下。

第一步：计算首项和末项的和 S，即 $S=1+1000=1001$。

第二步：计算 S 与项数的乘积 M，即 $M=S*1000=1001000$。

第三步：计算 M 除 2 的商 D，即 $D=M/2=1001000/2=500500$，得到最后结果 SUM＝500500。

比较算法一和算法二，在算法一中共用 500 次加法运算和 1 次乘法运算，得到正确结果，算法二中用 1 次加法、1 次乘法、1 次除法，也得到了正确结果。如果用计算机分别实现以上算法，由于 1 次乘法(除法)用时相当于 4～5 次加法用时，所以算法一计算较慢，耗时较多；算法二计算较快，性能较好。

由以上例子可以看出：算法是被精确定义的一组规则(执行序列)，这组规则明确规定先做什么，再做什么，并能判断在某种情况下完成怎样的操作，最终在有限的时间内执行有限的步骤后获得结果。规则不同，算法就不同，由此引起的算法性能也不同。

3.1.2 算法的特性

算法反映了求解问题的方法和步骤，不同的问题需要用不同的算法来解决，同一个问题也可能有多种不同的算法。但是，一个算法必须具备以下特性。

1. 有穷性

一个算法必须在有限的操作步骤内以及合理的时间内完成，即表现为时间上和空间上的有穷性。

2. 确定性

算法中的每一个操作必须有明确的含义，不允许存在二义性。

3. 有效性(可行性)

算法中描述的操作都是可执行的，并能最终得到确定的结果。包括以下两个方面：

(1) 算法中每一个步骤可以被分解为基本的在有限时间内可执行的操作步骤。

(2) 算法执行的结果要能够达到预期的目的，实现预期的功能。

4. 输入

一个算法有 0 个或多个输入，以刻画运算对象的初始情况，所谓 0 个输入是指算法本身定出了初始条件。

5. 输出

一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

3.1.3 算法的描述工具

描述算法有多种不同的工具，如自然语言、流程图、N-S 图、伪代码语言等，不同的算法描述工具在表达算法时有各自的优势，在特殊情况下，算法描述工具选择不当，将影响算法的质量。所以，设计算法前，应选择好合理的描述工具。下面简要介绍常见的算法描述工具。

1. 自然语言

自然语言就是人们日常使用的语言，如中文、英文、德文等。

例 3.1 两整数最大公因子的欧几里得算法可以描述为以下几步。

第一步：读入两个正整数 m 和 n（假设 $m>n$）。

第二步：求 m 和 n 的余数 $r=\mathrm{mod}(m,n)$。

第三步：用 n 的值取代 m，用 r 的值取代 n。

第四步：判断 r 的值是否为 0，如果 $r=0$，则 m 为最大公因子；否则返回到第二步。

第五步：输出 m 的值，即最大公因子。

文字形式的算法描述主要用于人类之间传递思想和智慧。当把算法用于人类和计算机之间传递智能时，文字形式的算法很难让计算机理解和执行。

2. 流程图

流程图是用一组规定的图形符号、流程线和简单的文字说明来描述算法的一种表示方法。常用的有传统流程图和 N-S 流程图两种。

例 3.2 欧几里得算法的传统流程图描述（见图 3-1）。

N-S 流程图：是美国学者于 1973 年提出的，通过顺序结构、选择结构、当型循环结构和直到型循环结构来描述具体算法。

（1）顺序结构。程序执行完语句 A 后接着执行语句 B，如图 3-2 所示。

（2）选择结构。当条件 P 成立时，执行语句 A，否则执行语句 B，如图 3-3 所示。

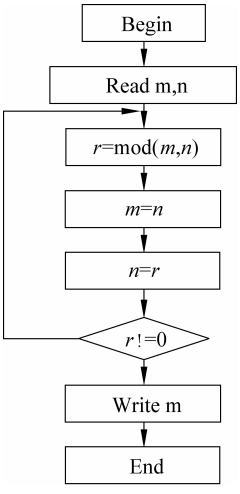

图 3-1 欧几里得算法

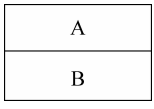

图 3-2 N-S 顺序结构

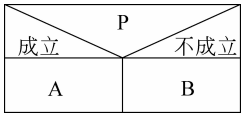

图 3-3 N-S 选择结构

(3) 当型循环结构。当条件 P 成立时,则循环执行语句 A,如图 3-4 所示。

(4) 直到型循环结构。循环执行语句 A,直到条件 P 不成立时为止,如图 3-5 所示。

例 3.3 欧几里德算法的 N-S 流程图描述(见图 3-6)。

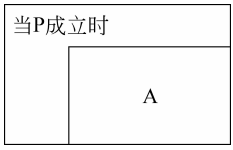

图 3-4 N-S 当型循环结构

图 3-5 N-S 直到型循环结构

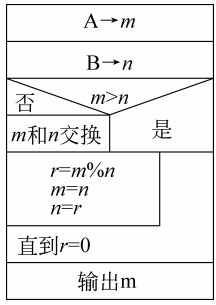

图 3-6 欧几里得算法(N-S 描述)

3. 伪代码

伪代码是一种介于自然语言与计算机语言之间的基于文字和符号的算法描述方法。其基本语句与计算机高级语言的语句非常接近,可以很方便地把伪代码形式的算法转变为计算机可以直接理解和执行的计算机高级语言程序。

例 3.4 欧几里得算法,用伪代码描述,算法如下:

(1) $r=m\%n$。

(2) 循环直到 r 等于 0。

① $m=n$。

② $n=r$。

③ $r=m\%n$。

(3) 输出 n。

4. 程序设计语言(C 程序设计语言)

程序设计语言能直接被计算机编译执行,但抽象性较差,对设计者的语言要求高。以下是用 C 程序设计语言描述的 Fibonacci 序列求解算法和欧几里得算法。

例 3.5 求 Fibonacci 序列的第 n 项。

```
//算法 Fibonacci
    //输入序列的下标 n(n>=0)
    //输出 Fibonacci 序列的第 n 项
      int Fibonacci (int n)
        {
            int F0,F1,F;
            int i=2;
            F0=F1=F=1;
            while(i<n+1)
            {
                F=F0+F1;
```

```
            F0=F1;
            F1=F;
            i++;
        }
        return F;
}
```

例 3.6 用辗转相除法求解两正整数的最大公因子(欧几里德算法)。

```
//算法 GCD
    //输入两正整数 a,b(a>0,b>0)
    //输出 a 和 b 的最大公因子
        int GCD(int a,int b)
        {
        int r,t;
        r=b;
        if(a<b)
        {
            t=a;
            a=b;
            b=t;
        }
        while(r!=0)
        {
            r=a%b;
            a=b;
            b=r;
        }
        return a;
}
```

3.1.4 算法的设计策略

算法设计的任务是对各类具体问题设计高质量的算法,以及设计算法的一般规律和方法。常用的算法设计策略主要有分治法、动态规划法、贪婪法、回溯法和分枝界限法等。

1. 分治法

分治法是把一个大规模问题划分成几个子问题,再把子问题分成更小的子问题……直到最后子问题可以简单地直接求解,求出子问题的解后,再把这些子问题的解答组成整个问题的解答。

2. 动态规划法

动态规划法是当整个问题无法由少数几个子问题的解答组合得出,而依赖于大量子问题的解答,并且子问题的解答又需要反复利用多次时,就系统地列表记录各个子问题的

解答，据此求出整个问题的解答。其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。动态规划的思想是多种算法的基础，被广泛应用于计算机科学和工程领域。

3. 贪婪法

贪婪法是指每一步选择都采用当前看来可行的或最优的策略。这是一种最直接的方法，只是在一些特殊的情况下，贪婪法才能求出问题的解答。对于最优解的问题，贪婪法通常只能求出近似解。

4. 回溯法和分枝界限法

为了寻求问题的解答，有时需要在所有的可能性（候选集）中进行搜索，例如在寻求最优解的问题中，就常碰到这种情况。这时，须把各种候选对象组织成一棵树，每个树叶对应着一个候选对象，于是每个内部顶点就表示若干个候选对象（即在此顶点下面的树叶）。回溯法是从树根开始按深度优先搜索的原则向下搜索，即沿着一个方向尽量向下搜索，直到发现此方向上不可能存在解答时，就退到上一个顶点，沿另一个方向进行同样的工作。分枝限界法也是从树根开始向下搜索，不同的是，分枝限界法常常利用一个适当选取的评估函数来决定应该从哪一点开始下一步搜索（分枝），以及哪一点下方不可能存在解答，从而这点的下方不必进行搜索（剪枝）。评估函数选得好，就会很快地找到解答，选得不好，就可能找不到解答或者找到的不是最优解（有时它可以作为最优解的一个近似解）。

3.1.5 算法的评价

同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。通常情况下，算法的优劣主要从算法的时间复杂度和空间复杂度来考量。

1. 算法的时间复杂度（时间特性）

算法的时间复杂度是指执行算法所需要的时间。一个程序在计算机上运行时所需消耗的时间取决于程序运行时输入的数据量、对源程序编译所需时间、执行每条指令所需时间以及程序中语句重复执行的次数，其中最重要的是程序中语句重复执行的次数。通常，把整个程序中语句的重复执行次数之和作为该程序运行的时间特性，称为算法的时间复杂度，记为 $T(n)=O(f(n))$，其中 n 为问题的规模，$f(n)$ 为问题的规模的函数。

在实际的时间复杂度分析中，通常考虑的是当问题规模趋向于无穷大的情形，以此简化时间复杂度 $T(n)$ 与求解问题规模 n 之间的函数关系，简化后的关系是一种数量级关系。例如，当某个时间复杂度为 $T(n)=3n^5+2n^3$，则表明程序运行所需时间与问题规模 n 之间是成 5 次多项式关系。当 n 趋向于无穷大时，有 $T(n)/n^5=3$，表示算法的时间复杂度与 n^5 成正比，记为 $T(n)=O(n^5)$。

算法的时间复杂度有 $O(1)$、$O(n)$、$O(n^2)$、$O(n^3)$、$O(n^4)$、$O(\log_2 n)$、$O(n\log_2 n)$、$O(2^n)$、$O(n!)$、$O(n^n)$ 等，其中时间复杂度最好的算法是常数数量级的算法，多数情况下得到的是多项式复杂度，经过优化后，很多能达到对数级复杂度（$O(\log_2 n)$、$O(n\log_2 n)$），这是较为理想的复杂度。对于数量级等同于 $O(2^n)$、$O(n!)$、$O(n^n)$，当问题规模 n 很大

时，计算机几乎很难在可接受的时间内完成运算并得到理想的结果，所以设计算法时应尽量避免。

2. 算法的空间复杂度（空间特性）

算法的空间复杂度是指算法需要消耗的内存空间。一个程序在计算机上运行时所占的空间同样也是问题规模 n 的一个函数，称为算法的空间复杂度，记为 $S(n)$，其中 n 为问题规模。为简化空间复杂度的求解，同样引入符号“O”，用于表达空间复杂度和问题规模之间数量级关系。例如 $S(n)=O(n^3)$，表示算法的空间复杂度与 n^3 成正比。

3.2 数据结构基础

计算机是一门研究用计算机进行信息表示和处理的科学。这里面涉及两个问题：信息的表示和信息的处理。而信息的表示和组织又直接关系到处理信息的程序的效率。计算机的普及、信息量的增加、信息范围的拓宽使许多系统程序和应用程序的规模很大，结构又相当复杂。因此，为了编写出一个“好”的程序，必须分析待处理对象的特征及各对象之间存在的关系，这就是数据结构这门课所要研究的问题之一。另外，计算机解决一个具体问题时，还需要给出每种结构类型所定义的各种运算即算法，这是数据结构这门课所要研究的另一重要内容。本节介绍数据结构的基本知识。

3.2.1 基本概念

1. 数据和数据类型

数据是信息的载体，它能够被计算机识别、存储和加工处理，包括数字、字母、汉字、图形、图像、音频和视频，在计算机内部表示为数值型数据和非数值型数据两大类。数据类型是指具有相同数据域并可以实施相同操作（运算）的数据的集合，例如，高级程序设计语言中的整型、字符型等，都是基本的数据类型。

2. 数据项、数据元素和数据对象

数据项是数据不可分割的最小单位。数据项有名和值之分，数据项名是数据项的标识，用变量定义，而数据项的值是它的一个可能的取值。

数据元素是数据的基本单位，具有完整、确定的实际意义。在不同的条件下，数据元素又可以称为元素、站点、项点、记录等。数据元素一般由若干数据项组成。

数据对象又称数据元素类，是具有相同性质的数据元素的集合，是数据的一个子集。在某个具体问题中，数据元素都具有相同的性质，属于同一数据对象，数据元素是数据元素类的一个实例。对于一个学生管理系统来说，某大学所有学生的基本情况就是数据，所有本科生的基本情况、所有硕士生的基本情况可以看作是不同的数据对象。

3. 数据结构

数据结构是指相互之间存在着一种或多种关系的数据元素的集合，它清楚地表达了

数据元素本身、数据元素之间的关系，以及基于数据元素和相互关系的操作。概括起来表现为数据的逻辑结构、数据的物理结构及数据操作(运算)，称为数据结构的 3 个要素。

(1) 数据的逻辑结构。数据的逻辑结构是指数据元素之际的逻辑关系，根据逻辑关系的不同，通常可以分成三类基本结构。

① 线性结构。数据元素之间存在着一对一的关系，除了第一个元素只有后继，最后一个元素只有前驱外，其余数据元素都有一个前驱和一个后继。典型的线性结构有线性表、栈、队列等。

② 树形结构。数据元素之间存在着一对多的关系，如树、二叉树和森林等。

③ 图形结构。数据元素之间存在着多对多的关系，如有向图和无向图等。邮政路径、铁路交通图都是典型的图形结构。

(2) 数据的物理结构。数据的物理结构是指逻辑结构在计算机存储器中的表示。数据的物理结构不仅要存储数据本身，还要存储数据之间的逻辑关系，同时还得考虑数据运算及存储效率等。数据的物理结构主要有顺序结构、链表结构、索引结构和散列结构四大类。

① 顺序结构。顺序结构是把所有数据元素存放在一片连续的存储单元中，逻辑上相邻的元素存储在物理上也相邻的存储单元中，由此得到的存储结构称为顺序存储结构。高级程序设计语言中提供的数组类型就属于这种存储结构，其最大优点就是可以实现随机访问，缺点是必须预先分析出所需定义数组的大小。如果预先定义的大小远远超过实际使用的大小，将造成内存空间的浪费；如果估计数组的最大个数小于实际使用的是数据元素个数，将导致程序无法正常运行。

② 链表结构。逻辑上相邻的数据元素不要求所占据的存储单元的物理位置相邻，元素间的逻辑关系通过附加的指针实现，这种存储结构称为链式存储结构。其优点是内存资源的使用合理，可能浪费的空间较少，缺点是操作的实现比顺序存储结构复杂。

③ 索引结构。针对每种数据结构建立一张索引表，每个数据元素占用表中一项，每个表项包含一个能够唯一识别一个元素的关键字和用于指示该元素所在存储单元的地址指针。

④ 散列结构。构造一个特定的散列函数，根据散列函数的函数值来确定数据元素存放的内存空间的地址。

(3) 数据运算。数据运算是指数据操作的集合。常见的数据操作包括数据的插入、删除、查找、遍历等。不同的数据结构具有不同的操作规则和方法。数据操作通常由计算机程序实现，也称算法实现。

3.2.2 常见的数据结构

1. 线性表

1) 线性表的定义

线性表是一种最简单最常用的数据结构，由有限个同类型的数据元素构成有序序列，元素之间存在一对一的关系，除了第一个元素只有直接后继，最后一个元素只有直接前驱

外，其余元素都有一个直接前驱和一个直接后继。

2）线性表的存储结构

在计算机中线性表可以有多种形式的存储结构，常用的有顺序存储结构和链式存储结构两种。

顺序存储结构是使用一批地址连续的存储单元来依次存放线性表的数据元素，如高级语言中的数组类型。采用这种存储结构实现对线性表的某些运算比较简单，如访问某个位置上的元素、求解线性表的长度等。但如果要实现插入、删除操作，则因为需要移动大量的数据元素而花费较多的时间。

链式存储结构的特点是使用不一定连续的存储单元来存放线性表。为了表示数据元素之间的逻辑关系，需要存储一个指示其直接后继的指针。整个线性表的各个数据元素的存储区域之间通过指针连接成一个链式的结构，因此又称为链表。采用链式存储结构不需要成片的连续存储空间，可以充分利用零碎的存储单元来存放元素。此外，还可以高效地实现插入、删除等运算。但由于需要存放额外的指针域，将会增加存储空间。

3）运算和实现

（1）遍历是指按某种方式，逐一访问线性表中的每一个元素，并执行读、写或查询等操作。

（2）查询是指在线性表中，按照查询条件，定位数据元素的位置。一般分为按值查询和按位置查询两种。

（3）插入是指在保持原有存储结构的前提下，根据插入要求，在适当的位置插入一个元素。

对于顺序存储的线性表，插入元素之前，要确保足够的存放空间，在满足插入条件的前提下，对第 i 个位置进行插入时，需要将 $n-i$ 个元素分别向后移动一个位置，然后在第 i 个位置处插入新的元素，同时线性表的长度增 1；对于链式存储结构的线性表，找到插入位置后，通过修改指针的指示方式来完成数据元素的插入。

（4）删除是指在线性表中找到满足条件的数据元素并删除。如果线性表为空，则删除操作无效。对于顺序存储的线性表，删除第 $i(i>=0$ 且 $i<n)$ 个元素时，通过将第 $i+1$ 到第 n 个元素依次向前移动一个位置的方式实现，同时线性表长度减 1；对于链式存储的线性表只要修改相关指针的指向即可。

2. 栈

1）栈的定义

栈是操作受限的线性表，即栈中规定只能在表的一端（表尾）进行插入或删除操作。该表尾称为栈顶（top）。设栈 $S=(a_1,a_2,a_3,a_4,\cdots,a_n)$，$a_1$ 是最先进栈的元素，称为栈底元素，a_n 是最后进栈的元素，称为栈顶元素。栈中元素按 $a_1,a_2,a_3,a_4,\cdots,a_n$ 的顺序进栈，而退栈的第一个元素是栈顶元素 a_n。即进栈和退栈操作是按照“后进先出”（last in first out，LIFO）的原则进行的。

2）栈的存储结构

在计算机中栈可以采用多种形式的存储结构，常用的有顺序存储结构和链式存储结构两种。一般情况下采用顺序存储的方式，即使用一个连续的存储区域来存放栈元素，并

设置一个指针 top，用来指示栈顶的位置，进栈和退栈只能在栈顶进行。

图 3-7 给出了栈顶指针 top 与栈内元素之间的关系，其中表示了栈的初始状态（空栈），栈元素 A、B、C、D 依次进栈以及 D 退栈的过程。

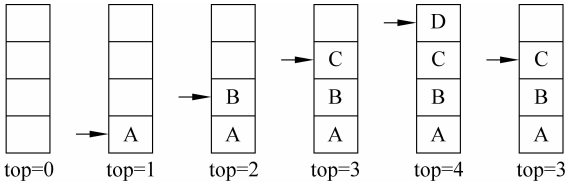

图 3-7　入栈、出栈过程

3）运算和实现

栈的基本运算主要有入栈、出栈、取栈顶元素和判空等操作。

（1）入栈，也称压栈，是在栈顶添加新元素的操作，新的元素入栈后成为新的栈顶元素。由于栈的大小是有限的，入栈时必须保证栈有足够的存储空间来存放新的元素；否则，会产生溢出，称为上溢。

（2）出栈，也称退栈，是将栈顶元素从栈中退出并传递给用户程序的操作，原栈顶元素的后继元素成为新的栈顶。出栈时必须保证栈内数据不空；否则，也会产生溢出，称为下溢。

（3）取栈顶元素，取得栈顶元素的值，栈中元素不变。

（4）判空，检查栈内数据是否为空，返回结果为一个逻辑值。如果栈顶和栈顶重合，说明栈为空，返回真值。

3. 队列

1）队列的定义

队列也是一种受限的线性表。与栈不同的是，在队列中规定只能够在表的一端（队尾 rear）进行插入，而在另一端（队头 front）进行删除操作。设 $Q=(a_1,a_2,a_3,a_4,\cdots,a_n)$，$a_1$ 为最先进入队列的队首元素，a_n 为最后进入队列的队尾元素。队列中元素按 $a_1,a_2,a_3,a_4,\cdots,a_n$ 的顺序进入，而退出队列的第一个元素是栈顶元素 a_1。即入队和出队操作是按照“先进先出”（frst in first out，FIFO）的原则进行的。这和日常生活中的排队是一致的。入队和出队操作的示意图如图 3-8 所示。

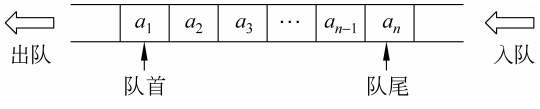

图 3-8　入队、出队过程

2）队列的存储结构

由于队列的数据结构变化较大，如果使用顺序存储结构，其中的数据要频繁地移动。因此，队列通常采用链式存储结构，用链表表示的队列称为链队列。一个链队列需要设置两个指针（队首指针和队尾指针），分别指向队列的头和尾，从而方便出队和