第1章 绪论

计算机是一门研究信息表示和处理的科学。信息表示包括组成信息的元素之间的相互关系(逻辑顺序)和信息元素在计算机中的存储方式(物理顺序);信息处理是根据解决实际问题的需要对信息加工计算的过程。

在计算机领域中,一般所讨论的"计算"有别于数学概念中的一般计算,通常将计算又称为"运算"或"操作"。运算或操作的内涵不仅包括传统意义上的四则运算和各种函数运算(公式化运算),而且还包括数据存取、插入、删除、查找、排序和遍历等运算。

从第一台电子计算机问世以来,计算机的应用主要包括两个方面:数值计算和非数值计算。计算机发展的初期,计算机主要为数值计算服务,其特点是计算过程复杂,数据类型相对简单,数据量相对较少。随着计算机的应用深入到各个领域,应用方面不再局限于数值计算的应用,更多地表现为非数值计算应用,非数值计算应用的特点表现为计算过程相对简单,数据类型相对复杂,数据的组织排列从某种意义上决定着非数值计算应用的有效性。

数值计算主要是指对一个或一组数据进行较复杂的四则运算、函数运算或迭代运算等。运算过程表现为数据处理的深入性,一般运算的原始数据对象较少。数据类型一般是数值型数据(整型、浮点型)。数值计算的程序设计主要围绕程序设计技巧,是典型的以程序为中心的设计过程。

程序设计中的非数值计算主要是指对类型较为复杂的大量数据进行内在联系的分析,根据处理的需要,合理地将数据按一定结构顺序进行组织存储,并完成对数据处理的程序描述。在非数值计算中,所处理的数据对象的类型一般较为复杂,通常是描述一个实体的若干个属性值的集合(结构类型或记录类型)。另外,数据是大量的,由于数据的大量性,所以移动全部或部分数据将消耗大量的时间或空间。还有,对数据进行较复杂的四则运算、函数运算等相对较少,较多地是对数据进行"管理"运算,如存取、查找、排序、插入、删除、更新等。

解决非数值计算问题,仅仅依赖程序设计的技巧已经无法达到提高效率的目的,必须对这些被加工数据的组织形式加以研究,找出最佳的数据组织形式,并与好的程序设计技巧相配合,才能达到提高效率的目的。所以,非数值计算问题是以复杂的数据为中心,研究数据的合理组织形式,设计出基于合理数据组织结构下的高效程序。

1.1 什么是数据结构

数据结构是随着计算机科学的发展而建立起来的围绕非数值计算问题的一门学问,而非数值计算是先于计算机科学存在的学问。在处理非数值计算问题时,首先要建立问题的数据模型,然后设计相应的算法。数据模型包含数据的组成结构,数据间的关联方式,对数

据实施相应运算后数据组成结构的完整性。数据组成结构的完整性是指不因对数据运算而改变数据模型的性质。运算方法本身是在保证数据组成结构完整性的前提下，以相同规律进行的。

1.1.1 数据结构相关事例

为了说明什么是数据结构，下面先讨论现实生活中的几个例子。

问题 1：电话号码簿和字典的使用问题。

当用户拿起一本厚厚的电话号码簿，查找自己需要的单位或个人的电话号码时，一定是从电话号码簿的分类目录开始，查找相应的大类别，然后根据所查找到的大类别后面指定的页码，翻到大类别的起始页，再从特定的大类别中查找小类别，从检索到的小类别下面顺序地找到用户所要的单位或个人的电话号码，如图 1.1 所示。

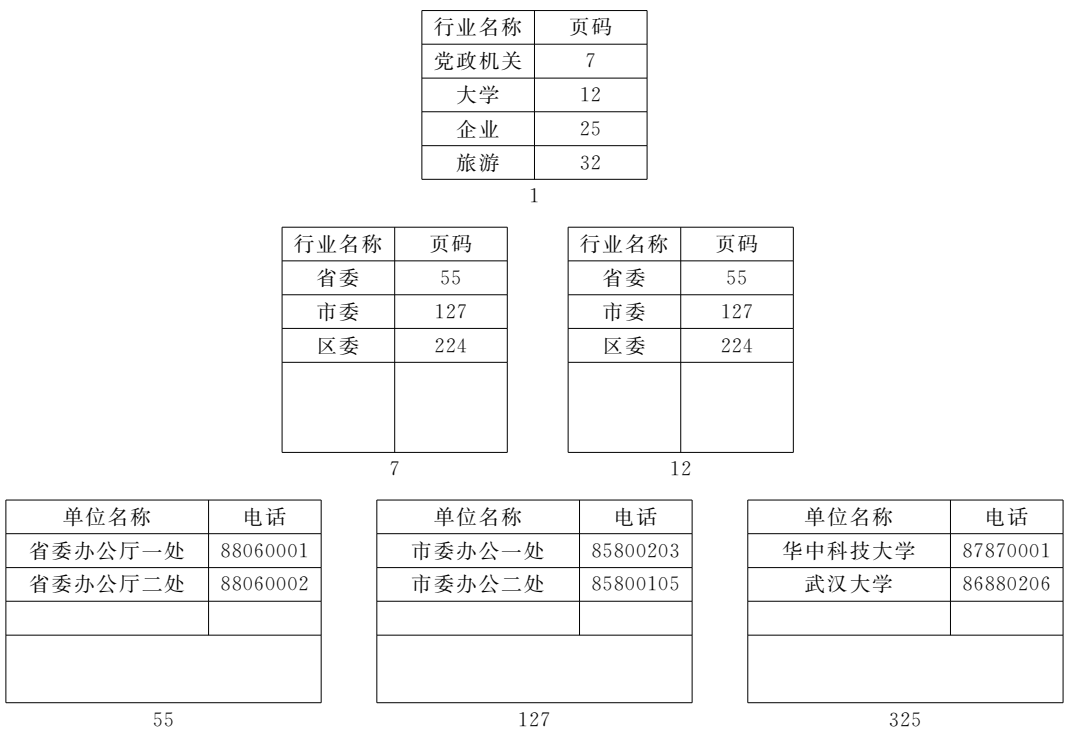

图 1.1 电话号码簿

大、小类别可以看作电话号码簿的目录或索引。如果电话号码簿缺少类别索引，而是按照电话安装先后的顺序进行排列或毫无规律地排列，用户会使用这样的电话号码簿吗？如果电话号码簿只记载着一个部门的十几部或几十部电话，还需要分类吗？

问题就是数据量的多少。少量数据的查找无须考虑数据的组织形式，而大量数据如果没有合理的组织形式，查找过程就只能顺序地进行，从而导致用户无法接受。在合理的数据组织结构下，用户会按照从大类别到小类别，然后在确定的小范围内顺序检索到所需要的信息。我们将检索过程称为检索的算法，每次检索都是按照同一规律进行的。电话号码簿的变更不会改变其数据的分类结构和数据的检索算法。

可见，这个问题的数据模型是对数据的两级分类索引结构，相应检索算法是先大类，后小类，然后在小类中顺序查找。无论是检索还是变更都要保证数据模型本身的完整性。

类似地，当用户在一本厚厚的汉语字典中查找某一个汉字时，首先必须知道所使用的字典的编码方法，然后才能按照偏旁部首，或者拼音等相应的编码方法较快地查到所需要查找的汉字。用户能如此顺利地在几万个汉字中找到所需要的汉字，是因为字典中的每一个汉字都是按偏旁部首或者拼音的规律严格地安排在它应处的页行(编码)上。倘若不按某一规律，将几万个汉字任意安排，用户为了查找某一个汉字就不得不从字典的第一页开始逐页地查找了。

可以看出查找效率是与字典中汉字的安排规律，也就是数据组织形式密切相关的。自然，对于不同组织形式必须采用相应的查找办法才能达到提高效率的目的。

问题 2：车皮调度问题。

有若干个发往同一方向不同城市的货车车皮以随机的次序到达货站的进车道，如图 1.2 所示，整个货站由三部分构成：进车道、出车道和调度车道(中间多条车道)。

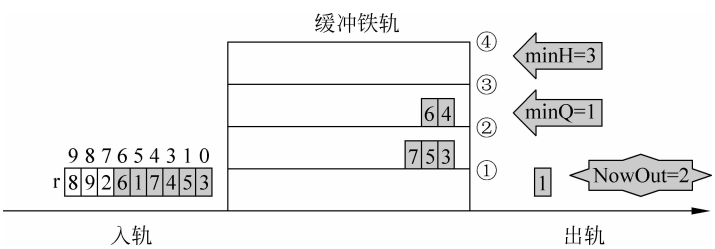

图 1.2 车厢调度转轨

由于货车的车皮从当前货站发往不同目的地，所以，货车发出本站时，列车的尾部所挂接的车皮应该是发往距本货站最近的车皮，这样可以保证到达某车站时从尾部“甩下”若干到站的车皮。

可问题是，不同的货运委托用户的车皮是以随机的次序到达货站的进车道的，为了便于说明，我们假设发往较远站点的车皮的编号比发往较近站点的车皮的编号要小，从图 1.2 中可以看到进车道上是不同的货运委托用户的车皮到达的次序。这些车皮发出本站前应调整它们的次序，调整为小编号在前，大编号在后的排列次序，即出车道上的次序。

为实现调整过程，货运站通过调度车道完成调度，调度车道由若干条缓冲铁轨组成，调度过程可描述为：

(1) 将入轨上第一个车皮任意选一个缓冲铁轨进入，一般选择第一个缓冲铁轨。

如：编号为 3 的车皮进入①号缓冲铁轨。

(2) 后面到达的车皮，如果其编号大于或等于已停放了车皮的每个缓冲铁轨上的最后一个车皮的编号，则选择与当前进入车皮编号最接近的缓冲铁轨进入。

如：编号为 5 的车皮进入①号缓冲铁轨，放在 3 号车皮的后面，因为 5＞3。

(3) 后面到达的车皮，如果其编号小于已停放了车皮的每个缓冲铁轨上的最后一个车皮的编号，则另选择一个没有停放车皮的缓冲铁轨进入。

如：编号为 4 的车皮进入②号缓冲铁轨，因为 4＜5。编号为 7 的车皮进入时，因为 7＞5 且 7＞4，选择接近的编号 5 所在的缓冲铁轨，编号为 7 的车皮进入①号缓冲铁轨，放在 5 号

车皮的后面。

(4) 如果下一个从入轨上进入的车皮编号正好比出轨上最后一个进入的车皮编号大1，则可将入轨上的这节车皮直接放到出轨上。

如：编号为1的车皮进入时，可以直接放到出轨上。

如此下去，所有车皮全部进入缓冲铁轨，可以看到缓冲铁轨上的车皮是有序排列的。

然后将缓冲铁轨上的所有车皮进入出车道，挂成一列车，挂接算法是：逐个地从缓冲铁轨中选出编号最小的车皮进入出车道，直至缓冲铁轨上的车皮全部进入出车道。

这一问题是数值计算无法完成的，它属于非数值计算问题。数据在调度过程中构成了有序“队列”，对于每个车皮的进入，采用同一运算方法进入缓冲铁轨。这就是数据结构中讨论的“队列”结构问题，从一头进入，从另一头出去。

问题3：电话通信线路问题。

某省各城市之间要架设电话通信线路，既要保证各城市间互通，又要使架设成本最小。就是数据结构中讨论的图结构的应用——最小生成树。

图1.3中给出了ABCDE 5个城市之间的距离，要实现各城市间互通，就要城市之间都有线路连接，要使成本最小，就要选择较近的城市之间架设线路，因此应该采用的架设结构如图1.4所示。

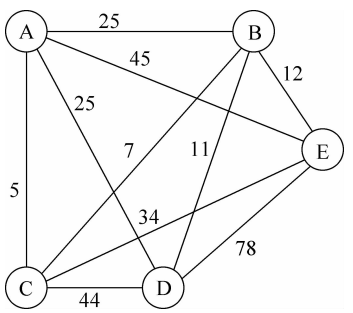

图1.3　城市连接及距离

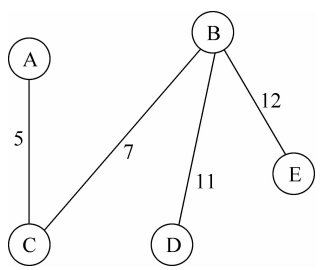

图1.4　最小生成树

数据的组织结构称为“图”结构，算法过程称为求取“最小生成树”。

可以看到算法过程就是保留连接线路中值最小的连接线路，且保证任何一城市都至少与另一个城市相连接。这样既保证了城市之间“连通”，又使“连通”成本最小。

这个问题中数据的组成结构是由顶点(城市)集合和边(线路)集合构成，求得的结果中，顶点(城市)集合没有发生改变，边(线路)集合减少了，但整个数据结构仍然是“图”结构。运算过程(以后再讨论)是一定的。

对上述3个问题归纳可以发现：

第1个问题中，数据可以看作由3个表存放，表与表之间的关系是由每个表项中的页码关联的；每个表中的数据之间的关系是由前后存放的次序进行关联的。检索时，前两个表之间利用页码指向跳跃查找，在第三张表中则顺序查找。这个问题涉及大量数据，所以在处理中要合理排列数据。

第2个问题中，数据在进车道时是按照先来排在前，后来排在后，不允许中间插入的原则形成一个队列的；在调度过程中，将进车道中的队列分散到不同的缓冲铁轨，每个缓冲铁

轨又是一个有序的子队列；最后，再形成一个有序的完整队列进入出车道。每个队列中元素以进入的先后进行关联。这个问题的处理中总是从队列的一端进入数据，从另一端移出数据。

第3个问题中，数据由两个集合组成，“顶点”集合中存放着顶点名称，“边”集合中存放着边的长度和边所关联的是哪两个顶点，两个集合中的数据可以是无序的。运算过程中找出较小的边，且保证保留的边使所有顶点间“连通”，但不形成环形路。这个问题的数据存放为两个集合。

从上面的3个问题可以看出，这些问题的模型都无法用数学的公式或方程或函数来解决，它们都是“非数值计算的问题”。

1.1.2 数据结构的定义

综上所述，可以这样给数据结构这门学科下个定义：数据结构就是研究计算机中大量数据存储的组织形式，定义且实现对数据的相应的高效运算，以提高计算机的数据处理能力的一门科学。

对该定义需要说明的是，数据的组织形式（结构）具有两个层面：

一个是与计算机本身无关的“逻辑组织结构”，简称“逻辑结构”，它的构成是由数据的值、数据之间的关联方式两个部分组成。如电话号码簿中“大类名、大类所指的页码”，“小类名、小类页码”，“单位名、单位的电话号码”是数据的值；数据间的关联是由大类所指的页码和小类页码共同实现的。

另一个是将具有逻辑组织结构的数据存放在计算机的存储介质上“物理组织结构”，简称“物理结构”。如电话号码簿中，每一张表内的数据可以在存储介质上以连续顺序存放，但表与表之间是在非连续的存储介质上分开存放的。

“运算”或“操作”是数据结构讨论内容的一个核心问题。不同实际问题具有不同的处理要求，所有的处理要求要进行事先定义，并用计算机的某种语言给予描述，这就是“算法设计”。

算法不仅要实现问题的要求，而且应该是高效地完成。低效的算法无法满足用户的需求或根本不能运用于实际。用一个高效的处理算法设计的程序结合高速运算的计算机可以最大程度地满足用户处理要求，而一个低效的处理算法设计的程序即使运用高速运算的计算机也不能满足用户的处理要求。

对实际问题的处理能力，不仅取决于计算机硬件本身的处理能力，而且更多地取决于对数据结构的合理组织以及相应处理算法的优劣。

1.2 数据结构的相关概念

为了更好地描述数据结构的内容，要用到许多术语，本节对相关的术语给出确切的含义，以便在今后的学习中能有统一的概念。

1.2.1 数据和信息

在计算机中，数据这个名词的含义非常广泛，可以认为它是描述客观事物的数字、字符

以及所有能输入到计算机中并能为计算机所接收的符号集合的总称。

数据是一个抽象的概念，它是一组符号的集合。单个符号是无法描述现实的，只有将多种符号进行组合，才可以描述现实。如单个字母或字符无法作为语言，字母或字符的组合才能形成语言。

数据是信息以某一类特定符号表示的形式，是计算机程序加工的对象。随着 IT 技术的发展，数据所能描述的信息越来越丰富，多媒体信息中的视频信号、音频信号经过专门设备的采集和转换后成为数字符号，形成计算机可存储并操作的数据。

1.2.2 数据元素

数据元素是数据集合中的个体，是数据组成的基本单位。数据通常是由若干个数据元素组成，数据元素是不可再分的最小单位。在数据存储组织中，它是基本的处理单位。

如图 1.5 所示，在商品信息的描述中，每个商品数据就是一个数据元素，商品数据元素的构成是商品名、商品编号、商品价格、商品数量，其中“商品价格”又可以再分为出厂价格、批发价格、零售价格。

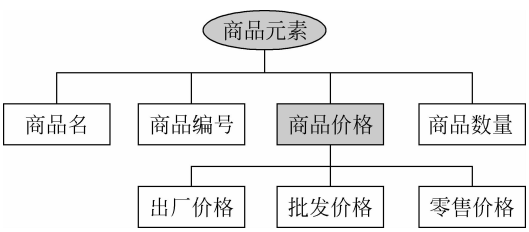

图 1.5 商品数据元素

构成商品数据元素的每个项目称为“数据项”(data item)，有的数据项是由单一的数据类型组成，称为原子项，而有的数据项又可再分为若干个子数据项组成，称为组合项。“商品价格”就是一个组合项。

数据元素是数据结构理论中较为抽象的概念，在具体描述中通常又表达为“记录”、“节点”或“顶点”。在计算机的高级语言中又可以有不同的表述，如 C 语言中定义为“结构类型”(structure type)，而在 PASCAL 语言中定义为“记录类型”(record type)。在数据库理论中定义为“元组”(tuple)。

在数据元素的多个数据项中能起标识作用的数据项称为关键字或关键码(key)。所谓标识作用即确定或识别数据元素的作用，如商品数据元素中，“商品编号”是唯一能标识某一商品数据的数据项，它就是商品数据元素的关键字。关键码如能唯一标识一数据元素，又称为主关键码，反之称为次关键码或次码。

1.2.3 结构类型

数据结构中讨论的结构类型分为两个层面，一是逻辑层面的数据结构，简称逻辑结构；另一个是物理层面的数据结构，简称物理结构。

1. 逻辑结构

逻辑结构描述数据元素与数据元素之间的关联方式，简称为关系，表示的是事物本身的

内在联系。逻辑结构又可以分为线性结构和非线性结构两大类。

线性结构的特点：数据元素之间存在前后次序，排在某个数据元素 b 前面的数据元素 a 称为 b 的直接前驱元素，而数据元素 b 则称为数据元素 a 的直接后继元素，对于某个数据元素，如果存在直接前驱元素或直接后继元素，则都是唯一的。线性结构中数据元素之间的正逆关系都是“一对一”的。线性结构又可再分为线性表、堆栈、队列等。

非线性结构的特点：数据元素不一定存在确定的前后次序，甚至是无序的，数据元素之间存在从属或互为从属的关系或离散关系。非线性结构又可再分为树形结构、图状或网状结构、纯集合结构。

在树形结构中，数据元素之间存在着“一对多”的关系。某个数据元素 a（节点）可能有多个分支，每个分支所连接的数据元素从属于数据元素 a（多个数据元素从属于一个数据元素）。

在图或网状结构中，数据元素之间存在着“多对多”的关系。如城市间的公路问题，每个城市数据元素与多个城市数据元素邻接，这些邻接数据元素相互之间不存在从属关系或理解为相互从属。

在纯集合结构中，数据元素具有“同属于一个集合”的关系。

纯集合结构和集合结构是不同的，纯集合结构中只有数据元素本身，而不存在数据元素之间的 “连接”关系；集合结构中除了有数据元素本身组成的子集外，还存在数据元素之间的 “连接”关系所组成的关系子集合。

所以说，所有逻辑结构类型都可以看作集合结构类型或集合结构类型的特例。例如，在集合结构中去掉一些“关系”，使每个数据元素连接为“一对一”，这时，集合结构就变成为线性结构。

2. 物理结构

物理结构也称为存储结构，是逻辑结构的数据元素在计算机的物理存储空间上的映像，映像不仅包含数据元素本身，而且包含着数据元素之间的关联方式，即关系的映像。

映像表现为两种方式：顺序映像和非顺序映像。

1）顺序映像

顺序映像是指数据元素在一块连续的物理存储空间上存储，物理存储空间只用于存放数据元素本身，数据元素之间的关联以两个数据元素存储的相邻关系来表示或通过某个函数来表示。或者说，利用数据元素在存储空间上的相对位置来表示数据元素之间的逻辑关系。

如一组成绩信息，每个数据由姓名和成绩两个数据项构成：

{{彭亮，97}，{王明，95}，{李智，90}，{刘丹，88}，{肖象，78}}

数据元素是按成绩从高至低的顺序排列，即按成绩从高至低关联，在物理存储空间上的存储映像如图 1.6 所示。

图 1.6 成绩信息的顺序映像

数据元素存储的空间是连续的，每个数据元素以相邻方式存储，相邻的两个数据元素左边元素的成绩大于右边元素的成绩。数据元素的前驱和后继关系是以数据元素存储的空间相邻性来表示的。如要查找第 i 名的成绩，则从第 i 个空间中可以得到。

再如，有一个下三角矩阵：

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 \\ B & C & 0 & 0 & 0 \\ D & E & F & 0 & 0 \\ G & H & I & J & 0 \end{bmatrix} \end{array}$$

在物理存储空间上存放为如图 1.7 所示，数据元素存储的空间是连续的，每个数据元素以相邻方式存储，数组元素按行优先法则存储。如要查找第 i 行，第 j 列（$1<i\leqslant 5$，$1\leqslant j<5$）数据元素，则从第 $((i-1)*(i-2))/2+j$ 个数据元素空间中可以得到，即数据元素关系函数为：

$$F(i,j)=((i-1)*(i-2))/2+j$$

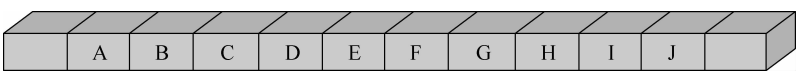

图 1.7　下三角矩阵的顺序映像

从上面两个问题中可以看出，数据元素顺序地映像在连续的存储的空间上，特别是矩阵从逻辑上看有行列之分，但在物理存储空间上无行列之分，逻辑上的行列关系是通过 F 函数来表示的，F 函数的值就是存储介质上某个空间的地址。

顺序映像的最大优点就是空间的利用率最高，但一旦要在中间插入数据元素或删除中间数据元素，就必须移动大量数据元素，这种运算在计算机中是相当耗时的。

2）非顺序映像

非顺序映像是指数据元素在物理存储空间上非连续地存储，物理存储空间不仅存放数据元素本身，而且为实现数据元素之间的关联，在每个数据元素存储的相邻空间中存储该数据元素关联的另一个或多个数据元素的起始地址。也可以说数据元素之间的关系是由附加“链接”或“指针”来表示的。

成绩信息的物理映像如图 1.8 所示。

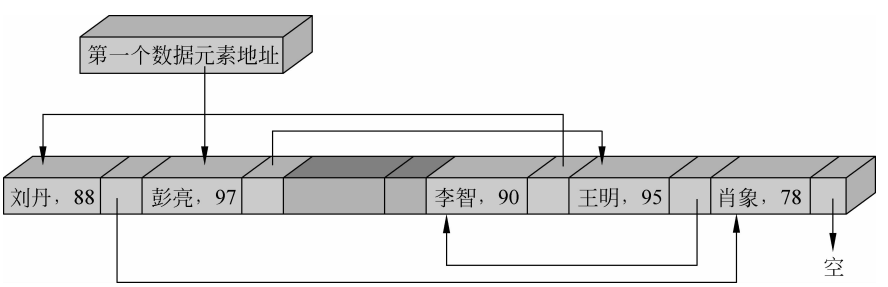

图 1.8　成绩信息的非顺序映像

非顺序映像存储结构中，数据元素的逻辑结构一般在物理空间上不是以物理空间的相邻来表现的，而是在每个数据元素本身所占用空间的相邻空间中，存放该数据元素所关联的另一个或多个数据元素的位置，通常将这个空间称为链地址空间。最后一个数据元素的链地址空间指向空地址。可以看出数据元素在物理存储上是离散的，且物理顺序不一定与逻辑顺序保持一致。

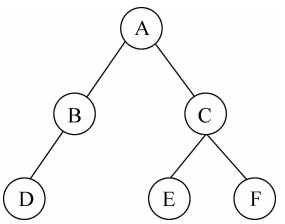

图 1.9 二分支的树逻辑结构

再如，有一个不超过二分支的树，如图 1.9 所示，数据元素 A 与数据元素 B、C 关联，可以认为 A 是 B 和 C 的父亲，B 和 C 是 A 的孩子，其他数据元素类似说明。该逻辑结构在物理存储空间上的映像可如图 1.10 所示。

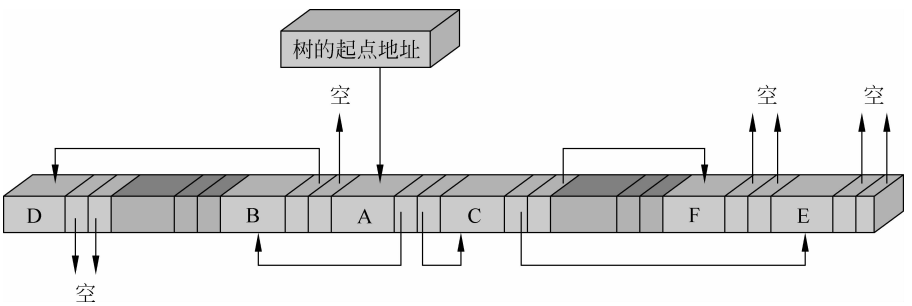

图 1.10 树的存储结构

这棵树在物理存储空间上的映像是离散的，物理空间上除存储每个数据元素本身外，还利用两个物理空间存储该数据元素所关联的孩子数据元素，没有孩子的数据元素的链地址空间指向空地址。

3. 数据域和链接域

1）数据域

数据域是物理存储空间中存储数据元素中数据值的空间，如成绩数据问题中存储姓名和成绩两个数据项，所占用的空间大小（字节数）依实际应用的数据元素中包含的信息量的大小而定。

2）链接域

链接域又称指针域，是非顺序存储映像时表示数据元素之间关系的地址存储空间，是额外的空间付出。一般地，在特定的计算机中，存放存储单元的地址所占用的空间大小（字节数）是一定的。在顺序存储映像结构中，链接域是不存在的，数据元素之间关系是以物理存储的邻接方式隐含表示的。

1.2.4 静态存储空间分配和动态存储空间分配

我们对数据结构的逻辑结构和物理结构已做了较详细的讨论，但未涉及物理存储空间的存储管理。存储管理不是研究某种数据结构，而是为满足各种数据结构对存储的不同要求，研究数据元素的空间分配、回收的方法和机制。

计算机的物理存储空间是有限的宝贵资源，对于每个空间的使用都要保证有效、合理。

计算机中的物理存储空间的使用包括两个方面：分配和回收，即数据元素对空间的使用（分配）和对数据元素使用过的空间的回收。

这就如同把计算机的所有物理存储空间看成饭店中的床位，某旅客入住时，只要饭店还有床位，就可以向管理饭店的管理员申请床位（分配），当旅客离开饭店时，就要到管理饭店的管理员处办理退床位（回收）手续。

1．静态存储空间分配和回收

一个团队在申请住房时，如果团队本身要求按相同性别及相同或相邻年龄逻辑顺序进行安排，那在分配床位时又可能采用两种方法。

饭店按团队要求的床位数一次性将连续的床位全部分配给这个团队，饭店不再具体安排某个人的床位，团队负责人再来安排每个人的具体位置。在保证按性别及年龄逻辑顺序进行安排的前提下，如果相同性别，相邻年龄的人保证入住在相邻的床位上，这是前面讨论过的顺序映像；如果相同性别，相邻年龄的人不保证入住在相邻的床位上，他们的逻辑关系只是在团队负责人的名单上是连续地排列，这就是前面讨论过的非顺序映像。

由于饭店按团队要求的床位数一次性将连续的（注意，保证连续性）床位全部分配给这个团队，饭店不再具体安排某个人的床位，所以，如果团队中有人中途离开，团队不去为少数人办理退床位手续，而是待整个团队离开时再一次性办理全部退房手续。

这种方法中最关键的是一次性申请连续床位，一次性退掉全部床位。在计算机中如果按一次性申请连续数据元素空间，一次性回收全部数据元素空间的模式完成空间的分配，而不处理零星的分配或回收，我们就将这种方式称为静态存储空间分配。在静态存储空间分配模式下，数据元素的物理映像可能是顺序映像，也可能是非顺序映像。

实际中，如果某用户使用 C 语言进行程序设计，程序中需要使用数组，在 C 语言中定义了一个数组，如果数组定义成功，这就表明一次性得到了连续数据元素空间。在所获取的数组空间中如何存储数据元素，是用户的工作，不是 C 语言所关心的事。C 语言只负责一次性空间的分配及数组使用完后一次性回收（数组空间释放）。

可见，静态存储空间分配模式下，数据元素的存储过程是分两步完成的，一是获取连续的物理存储空间，二是在获取的物理空间上进行物理映像。

2．动态存储空间分配和回收

如果饭店保证按性别及年龄逻辑顺序逐个地具体安排某个人的床位，这时，饭店可能有连续的床位进行安排，也可能没有连续的床位进行安排（分散安排）。

整个团队的逻辑顺序由饭店管理，这就是前面讨论过的非顺序映像。如果团队中有人中途离开或集体离开，该饭店管理员将为每个顾客逐个地办理退床位手续。这种方法中最关键的是逐个地申请床位，逐个地退掉床位。

在计算机中如果按逐个地申请数据元素空间，逐个地回收数据元素空间的模式完成空间的分配或回收，就称这种方式为动态存储空间分配。在动态存储空间分配模式下，数据元素的物理映像是非顺序映像，因为逐个获取的数据元素空间在物理上不一定是连续的。这种逐个地获取或回收反映了“动态性”。

实际中，如果某用户使用 C 语言进行程序设计，程序中需要若干个变量空间，在 C 语言中利用 malloc()函数或 new 申请变量空间（动态空间申请），如果某变量空间不再使用，则

利用 free()函数或 delete()释放变量空间(动态空间释放)。即 C 语言负责逐个空间的分配及逐个空间的回收。

可见动态存储空间分配模式下,数据元素的存储过程是一步完成的,即获取单个物理存储空间的同时,完成数据元素在物理空间上的映像。

对上面讨论的两种获取物理空间的模式进行分析后,我们引入一个存储池的概念,在计算机中将所有的物理存储空间称为存储池。对于操作系统而言,整个内存和外存都是存储池。C 语言就是在可管理的内存空间存储池中,完成数组的定义或数组的释放,变量空间的动态申请或变量空间的动态释放。

在计算机系统中,“操作系统”、“数据结构”、“编译原理”,“高级语言”等都涉及物理存储空间的分配,统称为存储管理。不同的是,每个学科的侧重点不同,概括地说,计算机系统中的存储管理分为以下 3 个不同的层次:

(1) 进程所需要的存储空间由操作系统分配,一旦进程结束,操作系统就回收进程所使用的存储空间。

(2) 数据元素存储空间由某个进程进行分配或回收。例如编译进程为变量、数组等进行存储空间的分配和回收。

(3) 数据元素存储空间由数据结构管理系统进行分配或回收。这类存储空间的管理问题是“数据结构”研究的范畴。

1.3 数据类型、抽象数据类型和数据结构

1.3.1 数据类型

抽象和具体是人们生活中常见的问题,就以“水果”为例,“水果”就是一种抽象,它是苹果、梨、西瓜等多种大品种的一种总结。事实上,“苹果”又是“富士”、“国光”、“金帅”等多种具体品种的再总结。所有的水果又具有共同的特性:植物类,高维生素含量,味道酸甜等。

再如,计算机的冯·诺依曼结构是由运算器、控制器、存储器、输入设备、输出设备构成的,这就是计算机结构的抽象。大型计算机的运算器和控制器结构与微型计算机的运算器和控制器结构是不同的,而不同微型计算机的运算器和控制器结构又是更具体的。无论是何种计算机,运算器和控制器的作用是相同的,只是微型计算机中的运算器、控制器在制作时被集成在一个芯片上。

事实上,一个系统是由多个对象构成的,其中往往存在多个具有共同特性的对象,表示为:$O_1,O_2,\cdots,O_n,\cdots$,这些对象都具有特性 $P_1,P_2,\cdots,P_m$,则具有特性 $P_1,P_2,\cdots,P_m$的对象便是 $O_1,O_2,\cdots,O_n,\cdots$对象的一个抽象。在计算机语言中,这样的抽象称之为“类”或“数据类型”。换言之,类或数据类型刻画了一组具有共同特性的对象。

数据类型(data type)的具体含义是,它描述了一组数据和在这组数据上的操作或运算及其操作或运算的接口。

一组数据是具有相同特性的数据集,例如,整数构成一个数据集,字符构成另一个数据集。数据集的个体可以由单数据项组成,也可以由多数据项组成。

而对具有相同特性的数据的操作或运算具有限制性。例如，整数可以进行加、减、乘、除、乘方运算；字符可以进行连接、求子字符串、求子符串在主串中的位置序号运算。

操作或运算的接口是约定在该数据类型上定义的一组运算或操作的各个运算名称，明确各个运算分别要有多少个参数及其参数的含义和顺序。

在对象式语言的程序中，程序中有一个或多个数据类型，在程序运行时，视需要创建该数据类型的各个对象，即具体实例。如在 C 语言中用数据类型 int 定义整型变量 a 并赋值，a 就是一个对象。

因此，数据类型是静态的概念，所创建的该数据类型的各个对象是动态的概念。

1.3.2 抽象数据类型

抽象数据类型(abstract data type，ADT)是指不涉及数据值的具体表示，只涉及数据值的值域、操作或运算，与具体实现无关，只描述操作或运算所满足的抽象性质的数据类型和接口。

抽象数据类型从数学意义上讲，是一种形式系统，抽象数据类型中的操作或运算是一种数学函数，表达的是一种操作或运算的功能，而用具体程序设计语言的语句来实现必然会以特定的方式表示。用具体程序设计语言实现时，首先要给出具体数据结构的表示，然后要在对象式语言中给出函数名或过程名及其相应参数，并给出操作或运算的具体语句序列。

我们也可以理解为底层数据类型是顶层抽象数据类型的具体实现。底层运算是顶层运算的细化，底层运算为顶层运算服务。为了将顶层算法与底层算法隔开，使二者在设计时不会互相牵制和影响，必须对二者的接口进行一次抽象。让底层只通过这个接口为顶层服务，顶层也只通过这个接口调用底层的运算。如果以 C 语言程序调用为例，也就是调用语句只要给出被调用函数的函数名、实参，无须了解被调用函数的实现编码，这就是一种顶层的“抽象”，而函数本身语句编码就是底层的具体实现。例如 C 语言中的字符串连接函数：

```
char * strcat( char * destin, char * source )
```

该函数就是相对于函数编码的抽象表达。调用时只要了解函数名是 strcat，函数的参数是 char * destin 和 char * source 的字符指针类型，连接结果是 * destin 所指的字符串，结果存放在该字符串中，* source 所指的字符串放在 * destin 前面。函数参数中，char * destin 参数必须出现在 char * source 的前面。而函数本身的语句编码一般用户可以不了解，对大多数用户来说是一个“黑箱”，语句编码就是底层的具体实现内容。这里需要说明的是，我们在这里用 C 语言函数例子所表示的还不是真正意义上的“抽象”，抽象数据类型是不依赖任何语言表达方式的，但可以用某种或某几种语言表示同一个抽象数据类型。

由此可见，数据类型描述了一组数据对象的共同特征，是对象式程序设计语言的基本成分，是抽象数据类型在对象式程序设计语言中的具体实现。可以说，抽象数据类型是数据类型概念的引申和发展。

使用抽象数据类型将给算法和程序设计带来很多优越性：顶层算法或主程序的设计与底层算法或子程序的设计被隔开，使得在进行顶层设计时不必考虑它所用到的数据和运算的具体表示和实现；反之，在进行底层数据表示和运算实现时，只要按照抽象数据类型定义的结构(名称和参数)实现，不必考虑它什么时候被引用。这样做，算法和程序设计的复杂性

降低了,有利于迅速开发出程序的原型,减少在开发过程中的差错。编出来的程序自然地呈现模块化,而且抽象的数据类型的表示和实现都可以封装起来,便于移植和重用,为自顶向下逐步求精和模块化设计提供一种有效的方法和工具。

1.3.3 数据结构、数据类型和抽象数据类型

数据类型和抽象数据类型,我们已经在前面较详细地讨论过,对于它们的相似性、相异性以及联系有了了解。那么它们与“数据结构”是什么关系呢?

“数据结构”作为计算机科学与技术领域的一门学科来讲,前面我们已经定义了,从狭义的角度理解,它用来反映一个数据的内部构成,即一个数据由哪些成分数据构成。

数据是按照数据结构分类的,具有相同数据结构的数据属同一类。同一类数据的全体称为一个数据类型,数据类型用来说明一个数据在数据分类中的归属,这个归属限定了该数据的变化范围和运算。不仅如此,数据结构还特别定义数据的逻辑结构和物理存储结构及其相应的物理存储映像。

简单数据类型对应于简单的数据结构,构造数据类型对应于复杂的数据结构。在复杂的数据结构里,允许成分数据本身具有复杂的数据结构。在构造数据类型中成分数据允许有不同的结构,因而允许属于不同的数据类型。

可见,数据结构是一个或多个数据类型的结构体,抽象数据类型的含义又可理解为数据结构的进一步抽象。即把数据结构和数据结构上的运算或操作捆在一起,进行“封装”。对于抽象数据类型的描述,除了必须描述它的数据结构外,还必须描述定义在它上面的运算或操作(过程或函数)。

1.3.4 抽象数据类型的三元组表示

抽象数据类型可以形式化地定义为三元组:

$$ADT=(Dset,Rset,OPset)$$

其中,Dset 表示数据元素集,Rset 表示数据元素的关系集,OPset 表示数据元素的基本操作集。以后为了全书的一致,约定以下面的格式描述各种抽象数据类型:

```
ADT 抽象数据类型名
{
  Dset: 【数据元素集描述】
  Rset: 【数据元素的关系集描述】
  OPset:【数据元素的操作集描述】
}
```

下面以二叉树来说明 ADT 说明:

```
ADT 二叉树抽象数据类型
 {
  Dset:  数据元素具有三个成员【数据值、左链接指针、右链接指针】
  Rset:  数据元素如果不空,则被分为根节点、左子树、右子树,每个子树仍是一个二叉树
  OPset:
        void  PreOrder (BinaryTreeNode * t)  二叉树的前序遍历
        void  PostOrder (BinaryTreeNode * t)  二叉树的后序遍历
        ⋮
 }
```

对于操作集中罗列的多个运算或操作是基本操作的集合，实际中可以补充或删减，对操作实现时，可能只讨论其中部分典型运算或操作。

1.4 算法及算法分析、算法描述

计算机是人们进行信息处理的工具，人们提出问题并利用计算机及其软件进行规划设计，解决问题。

从数学的角度而言，可以把解决问题看作定义函数，并利用定义的函数完成计算。函数是输入和输出之间的一种映射关系，函数的输入就是计算机程序中的参数，可以是一个值或多个值，这些值经过函数式的计算，将产生相应的输出，不同的输入可以产生不同的输出，但对于同样的输入，函数的计算输出一定是相同的。

同一个问题的解决函数可能不止一种，对同一个问题利用不同的函数运算，就是对相同的输入，经过不同的运算方法进行处理，但不同函数的运算结果应该是相同的。

1.4.1 算法和程序

1. 算法

算法是指解决问题的一种方法或一个过程。算法可以理解为函数的另一种表述，所以，一个给定的算法解决一个特定的问题。算法也可以描述为：

算法(algorithm)：算法是非空的、有限的指令序列，遵循它就可以完成某一确定的任务。它有 5 大特征。

- 有穷性：一个算法在执行有限步骤之后必须终止。
- 确定性：一个算法所给出的每一个计算步骤(通常是指算法描述中的下一步)必须是精确定义的，无二义性。
- 可行性：算法中要执行的每一个步骤都可以在有限时间内完成。
- 输入：算法一般有一组作为加工对象的量值，即对算法而言，就是一组输入变量。
- 输出：算法一般有一个或多个信息输出，它是算法对输入量的执行结果。

有穷性说明的是，解决问题的运算步骤是可以完全写出的，如果步骤是无限的，并无法全部写出，也就是无法将问题解决的步骤全部给出。

确定性说明的是，算法中所执行下一步是确定的，也可以进行下一步的选择，但选择过程必须是可控的和确定的。

可行性说明的是，算法中的每一步完成的时间是受限的，从另一个角度说，每一个步骤都应该是足够基本的操作。

输入说明的是，算法中的输入量是算法所需的初始数据，具有 0 个或多个。有的算法表面上看没有输入量(无参数)，而实际上输入量已经嵌入算法之中，这种算法通常是完成不受输入影响的特定工作。

输出说明的是，算法至少有一个输出，且要求算法给出的输出是所期望的结果。

2. 算法与程序差异

算法和程序是两个既相近又不同的概念，因此在现实中常常被人们不加区别地混用。

程序：是使用某种程序设计语言对一个算法或多个算法的具体实现。

同一个算法，可以使用不同的设计语言实现，所以可能有许多程序对应于同一算法。算法必须提供足够的细节才能转化为程序。算法是可终止的，但程序不一定，程序可在无外来干涉的情况下一直执行下去。程序可以既无输入又无输出信息。

下面的例子是一个程序，程序整体由三个部分组成：第一部分是包含说明和函数向前引用的说明；第二部分是主程序，由这部分控制整个程序执行的逻辑，其中就调用了一个算法 fact，用于求 n！的问题，并输出计算结果；第三部分是完成 n！具体求解的过程，也称为 n！的求解算法。

可以看出，算法只是程序中的一部分，在 C 语言中称为函数。算法是程序解决问题的核心。

```
#include <iostream.h>

int factorial (int n); //这是向前引用的函数说明

int main()
  {//这是主程序
    int         n = 6;
    int         result;
    result = factorial (n);
    cout <<" result = "<< result << endl;
    return 0;
}

int factorial (int n)
{//这是算法(函数),是程序的一部分
    int y = 1;
    for (int i = 1; i < n; i++)
    {
        y = y * i;
    }
    return y;
}
```

操作系统是一个程序，而不是一个算法，它可以在不发出停止命令的情况下无限地运行下去。但操作系统所完成的每一个功能就是要解决的一个问题，每一个问题由操作系统程序的一部分（即算法）来实现，且必须在有限步骤、有限时间内结束，并得到输出结果。

1.4.2 程序性能和算法效率

所谓程序性能，是指运行一个程序所需要的时间多少和空间大小。

测量程序性能的方法有两种：一种是通过直接执行程序，即实验法；另一种方法是分析法，分析程序编码的优劣。

在编制程序时，首先应该根据要解决的问题和数据性质选择一个恰当的数据组织结构和针对该组织结构的好的算法。相同数据组织结构下，对同一问题的不同算法要求具有相同的处理结果，但不同的算法处理的能力可能是不同的。

实际中，有些程序不能完全满足用户的全部要求，如有些用户限定程序运行时间的上限，程序运行一旦达到上限，就被强制结束，即运行时间过长；有些程序需求空间太大，系统资源无法满足，或随着运行过程，对空间的需求不断增大，结果最终导致空间被耗尽。

可见，程序是由算法组成的，程序的好坏是由算法的优劣所决定的。衡量程序的性能就是从空间复杂性和时间复杂性两方面衡量算法的性能。

那么，如何衡量一个算法的好坏呢？

衡量一个算法的好坏，从根本上讲，就是算法要有效利用计算机的资源。具体说，首先应确保算法满足上述 5 个性质，此外，通常还有考虑以下 3 个方面：

(1) 依据算法所编制的程序在计算机中运行时占有内存容量的大小(也要考虑占辅存容量的多少)，即空间特性。

(2) 依据算法所编制的程序在计算机中运算时所消耗的时间，即时间特性。

(3) 算法是否易理解，是否易于转换成任何其他可运行的语言程序以及是否易于调试。

1. 空间复杂性

程序所需要的空间主要由以下部分组成：

1) 指令空间

指令空间是指用来存储经过编译之后的程序指令所需的空间。编译之后的程序指令所需的空间的大小与编译器相关，不同的编译器所产生的指令空间是不同的，即使采用相同的编译器，所产生的程序代码的所需空间大小也可能不一样，例如，对编译代码是否优化所产生的代码所需空间就不一样。当然，使用不同类型的机器系统，所产生的代码所需空间也是不同的。

指令空间一般不是数据结构所讨论的问题。

2) 数据空间

数据空间是指用来存储所有常量和变量所需要的空间。数据空间分成三部分：

(1) 存储常量和简单变量所需要的空间。

如某程序中给定了整型常量 100，则该常量将占用一定的存储字节，程序中还定义了整型变量 k，用于循环控制，则系统也需要为变量 k 分配一定的存储字节。

(2) 存储复合变量所需要的空间。

这一部分空间又由两部分组成：一是数据结构定义的数据元素信息本身存储所需的存储空间，二是动态分配的存储空间。

3) 环境栈空间

环境栈用来保存函数调用和返回时需要的信息，包括返回地址、局部变量的值和参数的值。调用或递归的层次越深，所需要的环境栈空间就越大，这一部分的空间是可变部分。如求 n! 算法如采用递归方式，则递归层次说明如下：

```
Factorial(n)
  Factorial(n-1)
    Factorial(n-2)
      ⋮
    Factorial(1)
      Factorial(0)
```

2. 时间复杂性

一个程序在计算机上运算所消耗的时间主要取决于下述因素：

(1) 程序运行时所需要输入的数据总量消耗的时间；

(2) 对源程序进行编译所需要的时间；

(3) 计算机执行每条指令所需要的时间；

(4) 计算机关键指令重复执行的次数。

上面4个因素中，前两个取决于问题本身数据量的多少，第(3)个因素取决于计算机的硬、软件系统的客观条件。因此，一般把第(4)个因素作为分析算法时间效率的重点来讨论。

分析第(4)个因素主要可从两个方面估算：

(1) 找出一个或多个关键操作，确定这些关键操作所需要的执行时间；

(2) 确定程序执行步骤的次数，尤其是执行关键操作的次数。

程序执行过程中，数据的比较、变量的赋值(数据移动)、过程或函数的调用都是程序的关键性步骤，尤其是变量的赋值(数据移动)、过程或函数的调用步骤对程序时间复杂性影响较大。如果数据频繁移动，就会耗费大量时间在读写数据上。

执行关键操作的次数是决定程序时间效率的决定性因素。在程序中影响次数的程序结构主要是循环结构，所以在分析程序时间复杂性时重点分析循环过程及位于循环过程中的关键性步骤。

1.4.3 算法分析

算法分析不同于对程序分析直接执行的实验法，算法本身是程序的部分代码，所以在算法分析时只能分析算法中关键的部分，而忽略相对次要的算法操作步骤，把注意力集中在某些“关键”的操作上。不同的“操作”的难易程度是不同的，例如：

例1：

```
z1 = 0;
for (int y = 1;y < 10;y++)
{
    z1 = y + 10;
}
```

例2：

```
z2 = 0;
for (int y = 1;y < 10;y++)
{
    z2 = y * y + 100;
}
```

z1，z2的计算量是不同的，z1相对z2计算量较小。可是，在算法分析时，我们无论是计算z1或z2都认为是“一个操作步骤”，循环次数才是关键。这样做是为了突出分析的重点，忽略细节。

为此，引入语句频度(frequency count)的概念。所谓语句频度即“操作步骤”或“语句”重复执行的次数。为描述算法的时间复杂性，我们可以将复杂性用函数T(n)表示，n表示

算法中处理数据对象的数量或问题的规模的度量，如一算法时间复杂性是：

$$T(n)= 2n^3+3n^2+2n+1$$

当 n 足够大时，有 $T(n)\approx 2n^3$，或者 $g(n)=2n^3$。

因为，当 n 足够大时，$2n^3$ 的复杂度大大超过 $3n^2+2n+1$，或者说，T(n)数量级与 n^3 数量级相同。所以，当 n 足够大时，可以忽略复杂性函数中复杂性较低的部分，而只用复杂度高的部分表示。

进而引入渐进符号大“O”，用大写 O 字母表示函数 T(n)的上限函数，即当 n 足够大时的函数，记为：$O(g(n))= O(n^3)$。

下面给出几种典型的复杂性函数的表示(a,b,c 为已知数)：

常数函数：$O(g(n))= O(9+12)= O(1)$；

线性函数：$O(g(n))= O(a*n+b) =O(n)$；

对数函数：$O(g(n))= O((a*n*\log_2 n +b*n)= O(n*\log_2 n)$；

平方函数：$O(g(n))= O(a*n^2+b*n)= O(n^2)$；

指数函数：$O(g(n))= O(a^n+b*n^2+c*n)=O(a^n)$。

常数函数是指算法的复杂性与算法中处理数据对象的数量无关，比如：

$$T(n)=15$$

$$T(n)= 20056$$

两个 T(n)函数中，无论 n 的值如何，函数值始终为一常量，这时认为算法的复杂性是 O(1)，注意，不是 T(n)函数的常量值。

以上不同级别的复杂性函数存在着以下关系：

$$O(1)<O(\log_2 n)<O(n)< O(n*\log_2 n)<O(n^2)<O(a^n)$$

例如，两个 n×n 的矩阵相乘，其算法可描述如下：

```
void matrix-product(int a[][],int b[][],int c[][],int n);
  {
      for ( i=1;i<=n;i++)                              //重复次数:n+1
         for ( j=1;j<=n;j++)                           //重复次数:n(n+1)
             {
               c[i][j] =0;                             //重复次数:n²
               for ( k=1;k<=n;k++)                     //重复次数:n²(n+1)
                  c[i][j] =c[i][j]+a[i][k]*b[k][j];    //重复次数:n³
             }
  }
```

其中，每一语句的频度如上述算法注释所示。整个算法中所有语句的频度之和可约定作为该算法执行时间的度量，记作：

$$T(n)=(n+1)+n(n+1)+n^2+n^2(n+1)+n^3=2n^3+3n^2+2n+1$$

显然，它是矩阵阶 n 的函数，并且当 n 足够大时，则有 T(n) 数量级 $O(n^3)$。

再如，下面有三个简单的程序段。

(a)
```
x=x+1;
```

(b)
```
for (i=1;i<=n;i++)
      { x=x+1;}
```

```
(c) for (i = 1;i <= n;i++)
        for (j = 1;j <= n;j++)
            { x = x + 1;}
```

假定(a)中语句 x=x+1 不在任何的循环中,则语句频度为 1,其执行时间是一个常数,因此,其时间复杂性是常量级,即 O(1)。

在(b)中,同一语句 x=x+1 执行了 n 次,其频度为 n,其时间复杂性依赖于 n,所以,时间复杂性为 O(n)。

在(c)中,语句 x=x+1 执行了 n * n 次,频度为 n^2,其时间复杂性依赖于 n^2,因此,时间复杂性为 $O(n^2)$。

下面我们给出几个函数时间复杂性的曲线图,如图 1.11 所示,从图中可以看出,由于算法的复杂性不同,随着 n 的变化,算法的频度会相差很大,如果每步操作执行的时间是一定的,由于频度的不同,算法执行的时间也会有很大的差异。

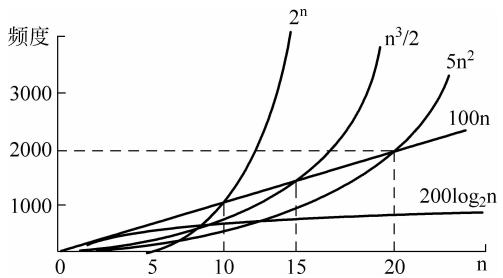

图 1.11 各种数量级的 T(n)

一个算法的时间复杂性是反映算法性能的重要指标。对于某一问题而言,算法时间复杂性的数量级越低,则说明算法的效率越高。也许有人说,现代计算机发展速度快得惊人,算法的效率无足轻重。然而,事实并非如此。在今天,问题需要处理的数据量越来越大,算法效率的高低对所能处理的数据量的多少有决定性的作用。当输入量急剧增加时,如果没有高效率的算法,单纯依靠提高计算机的速度,有时是无法达到要求的。

可以通过下述实例来说明这个问题。设有 5 个算法 A1、A2、A3、A4、A5,它们的时间复杂性如表 1.1 所示。给出了 1 秒钟、1 分钟和 1 小时内,这 5 个算法所能解决的输入量上界。从这里我们不难得到一个直观的概念:由于算法时间复杂性的数量级不同,它们在相同的时间里所能解决的问题量相差是极为悬殊的。

表 1.1 在某一时间内不同算法所能处理的输入量上限

算法	时间复杂性	1 秒钟能处理的最大输入量	1 分钟能处理的最大输入量	1 小时能处理的最大输入量
A1	n	1000	6×10^4	3.6×10^6
A2	$n\log_2 n$	140	4893	2.0×10^5
A3	n^2	31	244	1897
A4	n^3	10	39	153
A5	2^n	9	15	21

当计算机的速度成倍提高时,比如假定下一代计算机的速度比当代计算机的速度快 10 倍或 1 万倍,我们来分析一下上面这 5 个算法所能处理的输入量的大小有何变化。表 1.2 说明了由于计算机速度的提高给每个算法所带来的处理能力的改变情况。算法 A1 和 A2 在计算机的速度提高 10 倍或 1 万倍后,同一时间里所能处理的输入量几乎也增加

10 倍或 1 万倍；而算法 A3 和 A4 就差一些，最令人沮丧的是算法 A5，即使计算机的速度提高 1 万倍，算法 A5 在某一时间内所能处理的输入量不过比原来增加 13 个左右，真是廖廖无几。

表 1.2　计算机速度提高 10 倍和 1 万倍的效果

算法	时间复杂性	提高前单位时间内能处理的最大输入量	提高 10 倍后单位时间内能处理的最大输入量	速度提高 1 万倍后单位时间内能处理的最大输入量
A1	n	S1	10 S1	10 000 S1
A2	$n\log_2 n$	S2	对大的 S2 接近于 S2	当 $10\log_2 S2 > \log_2 9000$ 时超过 9000 S2
A3	n^2	S3	3.16 S3	100 S3
A4	n^3	S4	2.15 S4	21.54 S4
A5	2^n	S5	S5＋3.32	S5＋13.32

利用表 1.2 不难推出这样的结果：对于算法 A5 而言，一台高速计算机在 1 分钟内所能处理完的输入量只不过是比一台速度为它万分之一的低速计算机 1 分钟内所能处理完的输入量的两倍。

由以上例子可以看出，随着计算机应用的发展和要求处理的信息量越来越大，分析算法效率，设计出高效的算法越来越重要。

程序运行所占的存储量，同时也是问题规模的函数，它和时间复杂性类似，我们将以空间复杂性作为它的度量。

从主观上说，我们希望能选择一个既不占很多存储单元，运行时间又短且简明易读的算法。然而，实际上不可能做得十全十美，时间性和空间性往往是一对矛盾。许多程序中的数据经过压缩节省了存储空间，然而运行时解压缩的过程又需要额外的时间。再如，物理上顺序映像可以节省空间，但一旦进行插入或删除运算时，就造成数据元素的移动，时间效率就会大大降低。

往往一个看起来很简单的程序其运行时间要比复杂的程序慢得多，而一个运行时间较短的程序可能占用内存单元较多。因此，在不同的情况下应有不同的偏重选择，如果算法使用次数较少，则应力求简明易读，易于转换成上机程序；若算法转换成的程序使用次数较多，则就选择运行时间尽可能少的算法。若待解决的问题数据量大，而所使用的计算机存储容量又较小，则相应算法应着重考虑如何节省内存单元。在本书中，主要讨论算法的时间特性，针对某些问题再详细讨论空间特性。

1.4.4　算法描述

描述算法需要用一种语言来表达，这种语言可以是计算机程序设计语言，也可以是我们生活中使用的文字语言。为了便于阅读和转换为计算机上能够执行的程序，通常采用计算机中一种高级语言来描述算法，本书中将用类 C++语言(C++的一个子集)来描述。本书做如下使用规定。

1. 数据元素结构的表示用类型定义

如果对数据元素的处理不涉及数据元素的关键字数据项，则将数据元素描述如下：