第3章 算法与数据结构

3.1 绪 论

数据结构不仅是大学计算机专业的核心课程之一，也是其他相关专业主要选修的课程之一。数据结构是一门集数学、计算机硬件和软件于一体的综合性学科，它不仅是一般程序设计的基础，而且是实现操作系统、编译程序、数据库系统以及其他系统程序和大型应用程序的基础。算法是问题求解逻辑步骤的一种准确而完整的描述。计算机科学的重要基础是对算法的研究，而数据结构则是算法研究的基础。

著名的瑞士计算机科学家沃思(Nikiklaus Wirth)提出了一个模式：

程序＝算法＋数据结构

其中，算法是指对操作的描述，即操作的步骤；数据结构是指对数据的描述，即数据的类型和组织形式；程序是算法在计算机中的实现，算法和数据结构是程序的两大要素。程序设计的本质就是在分析具体问题后建立数学模型，然后针对数学模型设计出求解算法，再选择或设计一个良好的数据结构来存储相关数据，最后利用某种计算机语言编写程序解决问题。

数据结构作为计算机的一门重要学科，它主要研究三个方面的问题，如图 3-1 所示。

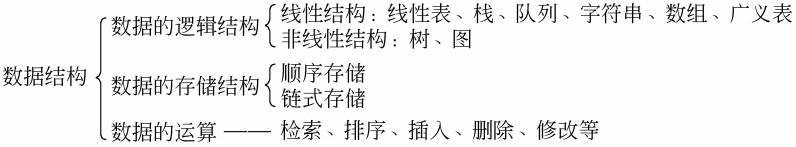


图 3-1 数据结构主要研究的三个方面问题

(1) 数据的逻辑结构：在数据集合中，各个数据元素之间的逻辑关系。

(2) 数据的存储结构：在处理数据时，各个数据元素在计算机中的物理存储关系。

(3) 在各种数据结构上进行的运算：检索、排序、插入、删除、修改等。

3.1.1 数据结构的基本概念

1. 数据

数据(Data)是指用符号来描述客观事物,它是所有能输入到计算机中并被识别、存储和处理的符号集合,是计算机化的信息。它包含数值、字符、图形、图像、声音等。

2. 数据项

数据项(Data Item)是描述数据的最小单位,能用来描述个体的具体信息。

3. 数据元素

数据元素(Data Element)是描述数据的基本单位,有时也称为记录、结点、顶点等。一个数据元素可以由一个或多个数据项组成。客观存在的一切个体都可以是数据元素,它在计算机程序中一般也作为个体进行考虑和处理。

4. 数据对象

数据对象(Data Object)是一类性质相同的数据元素的集合,是数据的子集。

5. 数据结构

数据结构(Data Structure)是指相互之间有关系的数据元素的集合。它由数据和结构两个部分组成,其中,数据是指数据元素的集合,结构是指数据元素之间关系的集合。也就是说,在计算机中的数据不是杂乱无章的,而是具有内在联系的数据元素的集合。所以在计算机中处理数据时,既要保存数据,又要保存数据之间的关系。

数据元素之间一般有如下 4 种基本结构(关系)。

(1) 集合结构:所有数据元素属于同一个集合,关系松散,如图 3-2(a)所示。

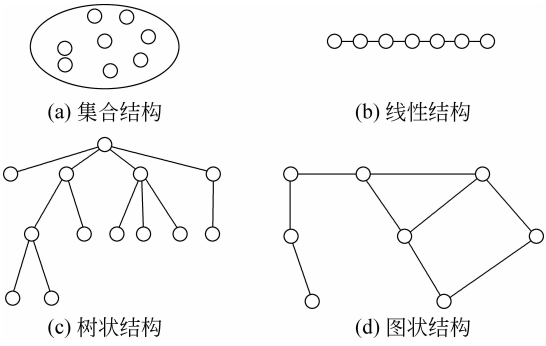


图 3-2　4 种基本数据结构示意图

(2) 线性结构:数据元素之间存在“一对一”的相邻关系,例如铁路春运购票队列,如图 3-2(b)所示。

(3) 树状结构：数据元素之间存在“一对多”的层次关系，例如高校组织机构，如图 3-2(c)所示。

(4) 图状结构：数据元素之间存在“多对多”的任意关系，例如交通图，如图 3-2(d)所示。

数据结构根据逻辑抽象描述方式和计算机内物理存储形式分为逻辑结构和存储结构两类。

1) 逻辑结构

逻辑结构(Logical Structure)是指使用数学模型来抽象地描述数据元素之间的逻辑关系。

一个数据的逻辑结构包括两个方面内容：

(1) 数据元素的信息；

(2) 数据元素之间的前后关系。

其中，数据元素之间的前后关系用前驱(或直接前驱)和后继(或直接后继)描述。

数据的逻辑结构可以用一个二元组表示：

$$\text{Data\_Structure}=(D,R)$$

其中，D 为数据元素的有限集合；R 为 D 上关系的有限集合。

例如，每周的数据结构用一个二元组表示为

$$\text{Week}=(D,R)$$

其中，$D=\{$周一，周二，周三，周四，周五，周六，周日$\}$；

$R=\{$(周一，周二)，(周二，周三)，(周三，周四)，

(周四，周五)，(周五，周六)，(周六，周日)$\}$。

数据的逻辑结构也可以用一个图形表示，例如家庭成员的数据结构的图形表示如图 3-3 所示。每个数据元素用中间标有数据元素值的方框表示，再用一条有向线段从直接前驱结点指向直接后继结点。家庭成员的数据结构的二元组表示为

$$\text{Family}=(D,R)$$

其中，$D=\{$母亲，儿子，女儿$\}$

$R=\{$(母亲，儿子)，(母亲，女儿)$\}$

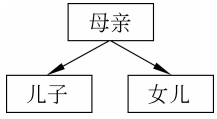


图 3-3　数据逻辑结构的图形表示

2) 存储结构(也称物理结构，Physical Structure)

存储结构是数据的逻辑结构在计算机内的存储形式，也称存储映像或内存映像。一种逻辑结构可能有多种存储结构。存储结构一般分为顺序存储结构和链式存储结构。

顺序存储结构的特点是借助于数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系，即逻辑上相邻的数据元素在物理存储地址上也相邻。

链式存储结构的特点是借助于指示数据元素存储地址的指针来表示数据元素之间的逻辑关系，即逻辑上相邻的数据元素在物理存储地址上不一定相邻。

另外还有索引存储结构和散列存储结构，但是它们都是通过顺序存储结构和链式存储结构复合而成的。

索引存储结构的特点是借助于数据元素的索引号来确定存储地址，检索速度快，但是附加了索引表的存储空间。

散列存储结构的特点是借助于数据元素的值来确定存储地址，检索速度快，但是若没有使用好散列函数可能会造成数据元素单元的冲突而需要附加存储空间。

由于数据元素在计算机存储空间中的物理位置关系可以与数据元素的逻辑关系不同，所以在描述数据结构时，数据的逻辑结构和数据的存储结构是数据结构不可分割的两个方面。一方面，数据的逻辑结构是数据结构的抽象；另一方面，数据的存储结构是数据结构的具体实现。一个算法的设计取决于数据的逻辑结构，而一个算法的具体实现则取决于数据的存储结构。在进行数据处理时，一种数据的逻辑结构可以依据需要采用多种不同的存储结构，如顺序存储结构、链式存储结构、索引存储结构和散列存储结构等，只要能反映出所要求的逻辑关系即可。但是，选用其中最合适的存储结构对于提高数据处理的效率是非常重要的。在实际应用中，不同的存储结构对算法的时间性能和空间性能等都有很大影响，即使是相同的存储结构也可以使用不同的算法来实现。

3.1.2 算法

用计算机解决实际问题，首先要给出解决问题的算法，再根据算法来编写程序。

1. 算法的定义

算法(Algorithm)是问题求解逻辑步骤的一种准确而完整的描述。对于一个实际的问题来说，若通过一个计算机程序，能在有限的存储空间中运行有限的时间后得到正确的结果，则这个问题就是算法可解的。程序可以作为算法的一种描述，它通过使用一些编程语言(例如 C 语言)来描述算法。同一个算法如果采用不同的编程语言能够编写出不同的程序，并且程序中一般还要考虑一些与算法无关的问题，如在编写程序时要考虑计算机运行环境的限制等。所以，算法不等于程序，程序的编写也绝不可能优于算法的设计。程序设计人员必须学会设计算法，并能根据算法利用高级语言编写程序，输入计算机中执行才能让计算机为人类服务。

描述算法的工具有自然语言、传统流程图、N-S 流程图、伪代码、类计算机语言、计算机语言。各种描述算法的语言在对问题的描述能力方面存在一定的差异。例如，自然语言灵活但不严谨；计算机语言虽然严谨，但是由于语法方面的限制显得不灵活；类计算机语言具有高级语言的一般语句，但是抛弃了高级语言中某些细节，便于把注意力集中在算法处理步骤的描述上。

2. 算法的基本特性

一般来说，一个算法应具有以下几个基本特性。

(1) 有穷性(Finiteness)：一个算法应包含有限的操作步骤。即一个算法必须在执行有穷步骤之后结束，并且其中每一步骤都可以在有穷时间内完成。例如，算法中若存在数学中的无穷级数时只能取有限项计算。

在实际应用中，有穷性也是指在合理的时间范围之内完成。如果一个算法用计算机执行需要一万年才结束，虽然算法是有穷的，但不是合理的。而程序与算法的主要区别是

程序可以无限地循环下去，例如操作系统中的监控程序在计算机启动以后就一直不停地监测操作者的操作动作直到停机。

(2) 确定性(Definiteness)：一个算法中的每一步骤必须有确定的含义，不能是含糊不清的，并且算法只有一个入口和一个出口。

(3) 可行性(Effectiveness)：一个算法应可以有效地执行，即算法描述的每一步骤都可通过已实现的基本运算执行有限次来完成。例如，算法中不能出现分母为零的情况。

(4) 输入(Input)：一个算法在执行时可能需要外部数据，也可能不需要外部数据，即一个算法有零个或多个外部输入。零个输入是指算法本身具有初始条件。

(5) 输出(Output)：一个算法的目的就是为了求解，即一个算法在执行完成后，一定要有一个或多个结果。或者说，一个算法至少要有一个输出，否则就失去了实际应用意义。当然，算法的输出是指一个算法得到了结果，并不一定是利用打印机进行输出。

3. 算法的基本要素

一个算法有如下两个基本要素。

(1) 算法中对数据对象的运算和操作：算法中基本的运算和操作包括算术运算、关系运算、逻辑运算和数据传输。

(2) 算法的控制结构：是指算法中各个操作之间的先后执行次序。一个算法的执行次序可以使用顺序、选择和循环三种基本结构组合而成。

4. 算法设计的基本方法

(1) 列举法：根据实际问题，列举出所有可能的情况。

(2) 归纳法：通过对特殊的情况进行归纳分析后找出一般的关系。

(3) 递推法：从已知的条件逐渐推出结果。它也属于归纳法。

(4) 递归法：先将复杂问题逐层分解(即回推)成最简单的问题，当解决了最简单的问题后，再沿着原来分解的逆过程逐步回归(即递推)以便解决复杂问题。

(5) 减半递推法：将复杂问题的规模减少一半，并且重复进行减半的过程。

5. 算法的评价

(1) 正确性：指根据算法编写的程序能得到满足问题要求的结果，它分为如下 4 个层次。

① 程序中不存在语法错误；

② 程序对于几组输入数据能得到满足要求的结果；

③ 程序对于苛刻的几组输入数据能得到满足要求的结果；

④ 程序对于所有合理的输入数据能得到满足要求的结果。

(2) 可读性：指算法首先应是便于理解，其次才是计算机可以执行。例如，使用的标识符要做到“见名知意”。有时为便于理解也可以加上注释。

(3) 健壮性：指输入不合法数据时算法能做出相应的处理，而不是陷入瘫痪。

(4) 高效率：指根据算法编写的程序有较快的运算速度。

(5) 低存储量：指根据算法编写的程序在运行时占用较少的存储空间。

6. 算法的复杂度

衡量算法的性能主要包括时间性能和空间性能等。大多数算法主要是对时间性能进行分析的，而算法的时间性能是以算法的时间复杂度来衡量的。

1) 算法的时间复杂度

算法的时间复杂度是指执行算法所需的计算工作量。由于一个算法在采用不同的语言、不同的编译程序以及在不同的计算机上运行时它们的效率都不同，所以不能使用绝对时间单位来衡量算法效率。算法的工作量应使用算法在执行过程中基本运算的执行次数来度量。例如，在两个矩阵相乘时，两个实数之间的乘法运算就作为基本运算，并且可忽略其中的加法运算。

算法执行的基本运算次数与问题的规模有关。例如，两个 30 阶矩阵相乘的基本运算次数一定大于两个 5 阶矩阵相乘的基本运算次数。

为了便于计算，算法的时间复杂度可以采用一种近似的形式，即数量级来表示。

算法的时间复杂度表示为

$$T(n) = O(f(n))$$

其中，O 表示数量级；n 是问题的规模。即表明算法的基本运算次数 $T(n)$是问题规模 n 的函数，并且 $T(n)$的增长率与 $f(n)$的增长率相同，$T(n)$是 $f(n)$的同阶无穷大。

例如：

```
for (i=1;i<=n;i=i+1)
    for (j=1;j<=n;j=j+1)
        { c[i][j]=0;
          for (k=1;k<=n;k=k+1)
            c[i][j]=c[i][j]+a[i][k] * b[k][j]; }
```

每层循环为 1～n 的三重循环，总的基本运算次数为 $n\times n\times n=n^3$，时间复杂度为 $T(n)=O(n^3)$。一般情况下只要大致计算出相应的数量级即可，不必精确计算出算法的时间复杂度，即可以通过对算法的时间性能的近似描述来简要了解算法的时间性能。

有些情况下，算法执行的基本运算次数与输入数据有关，此时可以从平均性态、最坏情况来进行分析。平均性态(Average Behavior)是指在各种特定输入下的基本运算的加权平均值；最坏情况(Worst-case)是指在规模为 n 时所执行的基本运算的最大次数。

2) 算法的空间复杂度

算法的空间复杂度是指执行算法所需存储空间的度量。它包括算法程序占用的存储空间、输入的原始数据占用的存储空间和执行算法所需额外占用的存储空间(如在链式存储结构中，既要存储数据又要额外存储链接地址以便来表示数据元素之间的关系)。

算法的空间复杂度表示为

$$S(n) = O(f(n))$$

其中，O 表示数量级；n 表示问题的规模。

3.2 线 性 表

3.2.1 线性表的基本概念

线性表是最简单的一种数据结构，它由一组数据元素组成。例如，一年的月份号(1,2,3,…,12)是一个线性表，表中的每个数据元素是一个整型数。再如，英文小写字母表(a,b,c,…,z)是一个线性表，表中的每个数据元素是一个小写字母。又如，表 3-1 所示的学生成绩表也是一个线性表，表中的每个数据元素(记录)都由学号、姓名、数学、外语、计算机等数据项组成。

表 3-1 学生成绩表

学　号	姓　名	数　学	外　语	计 算 机
001	赵一	91	83	75
002	钱二	82	75	83
003	孙三	73	67	91
004	李四	99	89	76
…	…	…	…	…

综上所述，线性表(Linear List)是由 $n(n\geqslant 0)$个类型相同的数据元素 $a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n$ 组成的有限序列。一般可以表示为

$$L=(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$$

其中，L 为线性表的名称，线性表中数据元素的个数 n 称为线性表的长度，当 $n=0$ 时称为空线性表。任意一个数据元素 $a_i(1\leqslant i\leqslant n)$也称为一个结点(Node)，它只是一个抽象的符号，其内容可以依据具体情况确定。同一个线性表中的数据元素必须具有相同的属性，数据元素可以是一个数据项，也可以由若干个数据项组成。由若干个数据项组成的数据元素也称为记录(Record)，由若干个记录组成的线性表称为文件(File)。

如果一个线性表非空，则 $a_i(1\leqslant i\leqslant n)$称为线性表的第 i 个数据元素，其中 i 称为 a_i 的位序(或序号)。$a_1,a_2,\cdots,a_{i-1}$称为 $a_i(2\leqslant i\leqslant n)$的前驱，$a_{i+1},a_{i+2},\cdots,a_n$ 称为 $a_i(1\leqslant i\leqslant n-1)$的后继，$a_{i-1}$称为 $a_i(2\leqslant i\leqslant n)$的直接前驱，$a_{i+1}$称为 $a_i(1\leqslant i\leqslant n-1)$的直接后继。在线性表中第一个数据元素 a_1 没有前驱，最后一个数据元素 a_n 没有后继。

在线性表中，数据元素的位置取决于它们的序号，数据元素之间的逻辑关系就是其邻接关系(如图 3-4 所示)，由于数据元素之间的相对位置是线性的，所以线性表是一种线性结构。

图 3-4 线性表的逻辑结构

非空线性表有如下特点。

(1) 有且只有一个根结点没有前驱结点。

(2) 有且只有一个终端结点没有后继结点。

(3) 除根结点和终端结点外，每个结点有且只有一个直接前驱结点和一个直接后继结点。

3.2.2 线性表的顺序存储及其基本运算

在计算机中存放线性表主要有两种存储结构，即顺序存储结构和链式存储结构。

把线性表中逻辑结构上相邻的数据元素依次存放在计算机内一组物理地址连续的存储单元中，即把逻辑关系上相邻接的数据元素存储在物理地址也相邻接的存储单元之中，这种存储结构称为顺序存储结构。在计算机中存储线性表最简单的方法是采用顺序存储结构，用顺序存储结构存储的线性表也称为顺序表。

顺序表具有的两个基本特点如下。

(1) 顺序表的所有数据元素所占的存储空间是连续的；

(2) 顺序表的各个数据元素在存储空间中是按照逻辑顺序依次存放的。

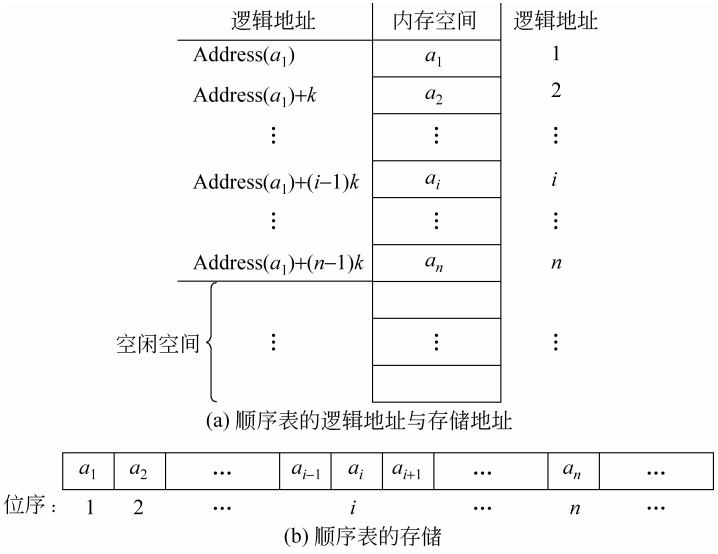
假设长度为 n 的顺序表 $(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$ 中每个数据元素所占的存储空间相同(假设都为 k 字节)，并且第 i 个数据元素 a_i 的存储地址用 $\mathrm{Address}(a_i)$ 表示，则相邻数据元素的存储地址可以表示为

$$\mathrm{Address}(a_i) = \mathrm{Address}(a_{i-1}) + k \quad (2 \leqslant i \leqslant n)$$

因此，顺序表中第 i 个数据元素 a_i 在计算机存储空间中的存储地址可以表示为

$$\mathrm{Address}(a_i) = \mathrm{Address}(a_1) + (i-1) \times k \quad (1 \leqslant i \leqslant n)$$

若知道顺序表的起始地址 $\mathrm{Address}(a_1)$ 和每个数据元素所占的字节数 k，则顺序表中任意一个数据元素都可随机存取。顺序表的存储结构是一个随机存取的存储结构，如图 3-5 所示。



(a) 顺序表的逻辑地址与存储地址

(b) 顺序表的存储

图 3-5 顺序表存储的两种示意图

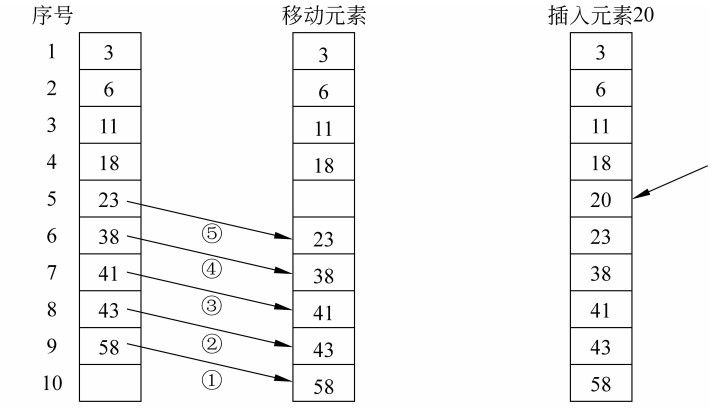
在使用高级语言(例如C语言)进行程序设计时,由于一维数组与计算机中实际存储空间结构类似,所以一般使用一维数组来表示线性表的顺序存储(即顺序表)空间。顺序表所需的最大存储空间有时不好预测,如果开始时所开辟的存储空间太小,可能在顺序表动态增长时会造成存储空间不够;但若开始时所开辟的存储空间太大,就有可能造成存储空间浪费。一般情况下,可以根据顺序表动态变化时的一般规模来决定。

1. 顺序表的插入运算

顺序表的插入运算是指线性表在顺序存储结构下的插入运算。它是在线性表的第$i(1\leqslant i\leqslant n+1)$个数据元素的位置之前插入一个新数据元素$x$,使长度为$n$的线性表$(a_1,a_2,\cdots,a_{i-1},a_i,\cdots,a_n)$变成长度为$n+1$的线性表$(a_1,a_2,\cdots,a_{i-1},x,a_i,\cdots,a_n)$。

通常情况下,若要在长度为n顺序表的第$i(1\leqslant i\leqslant n+1)$个数据元素之前插入一个新的数据元素,则需要将从最后一个(即第n个)数据元素开始,直到第i个数据元素之间共$n-i+1$个数据元素依次向后移动一个位置,即把位序为n、$n-1$、…、$i+1$、i的数据元素依次向后移动到$n+1$、n、…、$i+2$、$i+1$位置上,然后将新的数据元素x插入到新空出的第i个位置上。

例如,已知顺序表(3,6,11,18,23,38,41,43,58),若要在第5个位序数据元素之前插入一个新的数据元素20,则要把第9个位序(元素58)到第5个位序(元素23)之间的数据元素依次向后移动一个位置,再将新的数据元素20插入到第5个位序,如图3-6所示。



(a) 数据元素从后向前依次向后移动一个位置　　(b) 插入新的数据元素

图3-6　顺序表插入运算的前后变化

顺序表插入运算(即线性表在顺序存储结构下的插入运算)有如下几种情况。

(1) 在通常情况下,若在第$i(1\leqslant i\leqslant n+1)$个数据元素之前插入一个新的数据元素时,要将从最后一个(即第n个)数据元素开始,直到第i个数据元素之间共$n-i+1$个数据元素依次向后移动一个位置。

(2) 在特殊情况下,若要在顺序表第一个数据元素之前插入一个新的数据元素,则需要依次向后移动表中所有的n个数据元素;若要在顺序表最后一个(即第n个)数据元素之后(即第$n+1$个数据元素之前)插入一个新的数据元素,则不需要移动表中的数据元

素，只要在表的末尾后面增加一个新的数据元素即可。

(3) 在平均情况下，插入一个新的数据元素需要移动表中一半(即 $n/2$ 个)的数据元素。

由于数据元素的移动消耗很多时间，因此顺序表的插入运算的效率非常低，尤其在顺序表的长度较大的情况下更为严重。另外，在顺序表的插入运算中，若为顺序表开辟的存储空间已满，此时就不能再进行插入，否则会产生“上溢”错误。

2. 顺序表的删除运算

顺序表的删除运算是指线性表在顺序存储结构下的删除运算。它是在线性表中删除第 $i(1\leqslant i\leqslant n)$ 个位置上的数据元素，使长度为 n 的线性表 $(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$ 变成长度为 $n-1$ 的线性表 $(a_1,a_2,\cdots,a_{i-1},a_{i+1},\cdots,a_n)$。

通常情况下，若要在长度为 n 的顺序表的第 $i(1\leqslant i\leqslant n)$ 个位置上删除一个数据元素，则需要从第 $i+1$ 个数据元素开始，直到最后一个(即第 n 个)数据元素之间共 $n-i$ 个元素依次向前移动一个位置，即把位序为 $i+1,i+2,\cdots,n-1,n$ 的数据元素依次向前移动到位序为 $i,i+1,\cdots,n-2,n-1$ 的位置上。

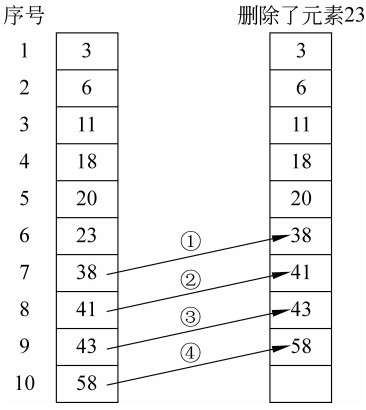


图 3-7　顺序表删除运算的前后变化

例如，已知顺序表(3,6,11,18,20,23,38,41,43,58)，若要删除在第 6 位序的数据元素 23，则要把第 7 个位序上的数据元素(38)到第 10 个位序上的数据元素(58)之间的数据元素依次向前移动一个位置，如图 3-7 所示。

顺序表删除运算(即线性表在顺序存储结构下的删除运算)的几种情况。

(1) 在通常情况下，若要删除第 $i(1\leqslant i\leqslant n)$ 个位置上的数据元素时，要将从第 $i+1$ 个数据元素开始，直到最后一个(即第 n 个)数据元素之间共 $n-i$ 个数据元素依次向前移动一个位置。

(2) 在特殊情况下，若要删除第一个数据元素，则需要依次向前移动表中所有其他的 $n-1$ 个数据元素；若要删除最后一个(即第 n 个)数据元素，则不需要移动表中的数据元素，只要删除表中末尾一个数据元素即可。

(3) 在平均情况下，删除一个数据元素需要移动表中一半(即 $(n-1)/2$ 个)的数据元素。

由于数据元素的移动非常消耗时间，因此顺序表的删除运算(即线性表在顺序存储结构下的删除运算)的效率非常低，尤其在顺序表的长度较大的情况下更为严重。另外，在顺序表的删除运算中，若为顺序表开辟的存储空间内容已空，此时就不能再进行删除，否则会产生“下溢”错误。

线性表顺序存储结构(即顺序表)的优缺点如下。

① 优点：逻辑关系上相邻的数据元素(结点)在计算机中存储的物理位置上也相邻，所以不需要为表示数据元素的逻辑关系而增加额外的存储空间，并且可以根据初始地址

随机存取线性表中任一数据元素；运算简单方便，存储空间的使用紧凑，占用了最少的存储空间。

② 缺点：除了线性表表尾的数据元素（结点）外，在其他位置进行插入、删除操作时需要移动大量的数据元素，平均需要移动表中约一半的数据元素，而大量数据元素的移动非常消耗时间；而且，由于占用连续的存储空间，所以只能进行预先静态分配，若开辟的存储空间太大，就有可能造成存储空间浪费，若开辟的存储空间太小，当分配的存储空间已满时，再插入数据就会发生“上溢”错误，不便于扩充；另外，由于只能使用相邻的整块存储单元，也可能会产生较多的碎片现象。

由于顺序存储结构相对简单，所以线性表的顺序存储结构适用于小线性表或者数据元素不常变动的线性表。又因为数据元素的移动消耗很多时间，导致顺序表的插入、删除运算的效率非常低，所以线性表的顺序存储结构不适用于大线性表或者数据元素经常变动的线性表。顺序表适合于主要进行查找而很少做插入和删除操作的数据。

3.2.3 线性表的链式存储及其基本运算

顺序表（即线性表的顺序存储结构）的优点是逻辑关系上相邻的数据元素在计算机中存储的物理位置也相邻，则可以根据初始地址来随机存取线性表中的任一元素。它的缺点是进行插入、删除操作时平均需要移动线性表中一半的元素，并且要进行预先静态分配存储空间，不便于扩充。

由于线性表的顺序存储结构不适用于大线性表或者数据元素经常变动的线性表，为了避免由于大量数据元素的移动而消耗处理时间导致顺序表的插入、删除运算效率低的缺点，线性表可以使用另外一种存储结构——链式存储结构。

使用一组任意的存储单元来存储线性表中的数据元素，这些存储单元在计算机内的物理位置可以是连续的，也可以是不连续的，这种存储结构称为链式存储结构。用链式存储结构来存储的线性表称为线性链表，简称链表（Linked List）。根据链接的方式把链表分为单链表、双链表和循环链表等。链表适合于频繁地进行插入和删除操作的数据。

1. 单链表

因为链式存储结构不要求逻辑关系上相邻的数据元素在计算机中存储的物理位置也相邻，所以为了表示数据元素（或结点）之间的逻辑关系，每个数据元素（或结点）的存储空间就被分为两个部分。其中一个部分用来存储结点的值，称为数据域；另一个部分用来存储指向直接后继的指针（或后继的地址），称为指针域。所有的结点通过指向直接后继的指针（或后继的地址）的连接按其逻辑顺序链接在一起而组成了链表，使得逻辑上相邻的结点的物理位置不一定相邻。除了数据域外，如果每一个结点只包含一个指针域，这样形成的线性链表就只有一个方向的链，则称为线性单链表，简称为单链表（Single Linked List），如图 3-8 所示。

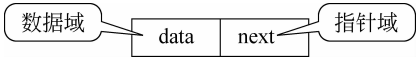


图 3-8 单链表的结点形式

其中，data 是数据域，用来存放结点的值；next 是指针域（也称链域），一般用来存放结点的直接后

继的地址(或位置)。

由于在单链表中每个结点的存储地址都保存在其直接前驱结点的指针域中,而第一个结点无直接前驱结点,为增强程序的可读性,在线性单链表中一般还增加一个指针指向链表的第一个结点(即存储第一个结点的地址),称为头指针(HEAD)。一个单链表可以用头指针的名字来命名。最后一个结点由于没有直接后继结点,则它的指针域值为空(用NULL、0 或 ∧ 表示),表示链表终止。由于任意一个结点的存取都必须从头指针出发,顺着链域逐个结点往下进行,所以单链表是顺序存取的存储结构,如图 3-9 所示。

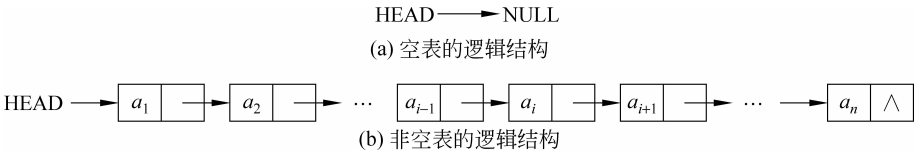


图 3-9　线性单链表的逻辑结构(不带头结点)

例如,线性单链表(a_1,a_2,a_3,a_4,a_5,a_6)采用链式存储,如图 3-10 所示。

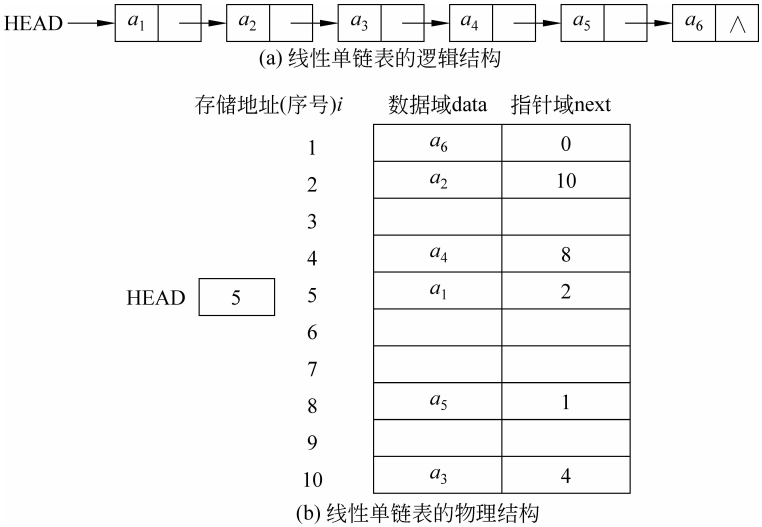


图 3-10　线性单链表(不带头结点)

链表的特点是通过每个结点的指针域(也称链域)将线性表中的所有结点按其逻辑顺序链接在一起的。通常情况下,各个结点的存储地址(或序号)不是连续的,并且在存储空间的位置关系与逻辑关系也不一致。

为了操作方便,有时可以在单链表第一个结点之前再增加一个结点,称为头结点。头结点的数据域为任意或依据需要设置,头结点的指针域存储指向第一个结点的指针(即第一个结点的存储地址)。此时,头指针 HEAD 指向头结点,而头结点指向单链表的第一个结点,如图 3-11 所示。

1) 建立线性单链表

(1) 头插法建立线性单链表:从空链表开始,每次申请生成一个新的结点,将读入的

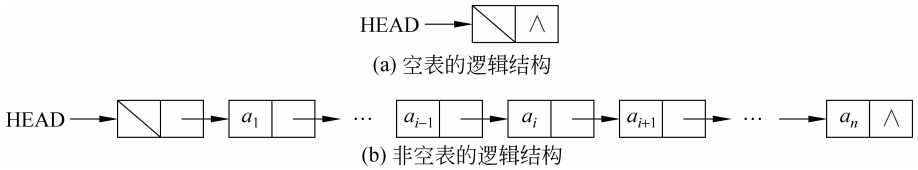


图 3-11　线性单链表的逻辑结构(带头结点)

数据存放到数据域,并设置指针域为空。然后将新结点插入到链表头结点的后面,即新结点的指针域存储原来头结点指针域存放的指针,再将头结点的指针域存储指向新结点的指针(或存储地址),重复以上操作,直到读入的数据为结束标志,如图 3-12 所示。

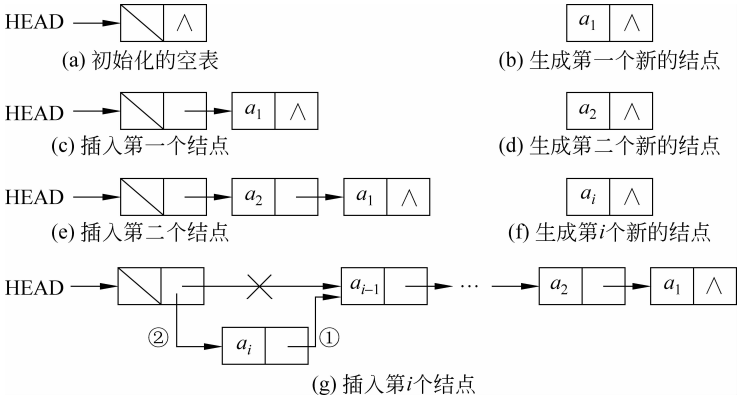


图 3-12　头插法建立线性单链表

使用头插法建立线性单链表时,结点的逻辑顺序与输入数据元素的顺序相反。

(2) 尾插法建立线性单链表:从空链表开始,每次申请生成一个新的结点,将读入的数据存放到数据域,并设置指针域为空。然后将新结点插入到链表尾结点的后面,即原来链表尾结点的指针域存储指向新结点的指针(或存储地址),此时新结点转变为新的尾结点,重复以上操作,直到读入的数据为结束标志,如图 3-13 所示。

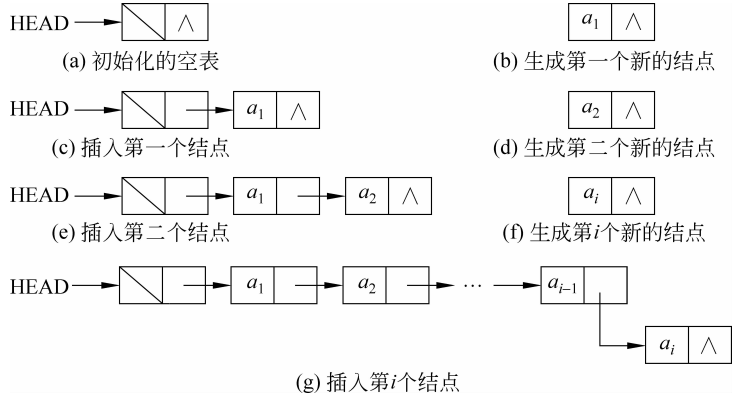


图 3-13　尾插法建立线性单链表

使用尾插法建立线性单链表时，结点的逻辑顺序与输入数据元素的顺序相同。

2）线性单链表的查找

为便于下面进行线性单链表的插入或删除操作，需要先查找到插入或删除的位置。

线性单链表的查找是指在线性单链表查找指定的数据元素。

（1）按照值进行查找：从线性单链表的头指针出发，顺着链表逐个将结点数据域的值和要查找的给定值作比较。若查找到有结点数据域的值等于给定值时，则返回首次找到的结点存储位置，否则返回 NULL。在实际应用中，为了便于操作，需要在线性单链表中查找包含给定值 x 元素的前一结点存储位置（可用指针 pre 指向），就可以在该结点后插入新的结点或者删除该结点后的结点。查找的结果如下。若单链表中存在要查找的给定值，则返回的(pre)为第一次找到的包含给定值 x 元素的前一结点位置；若单链表中不存在要查找的给定值，则返回的(pre)为单链表的最后结点位置，如图 3-14 所示。

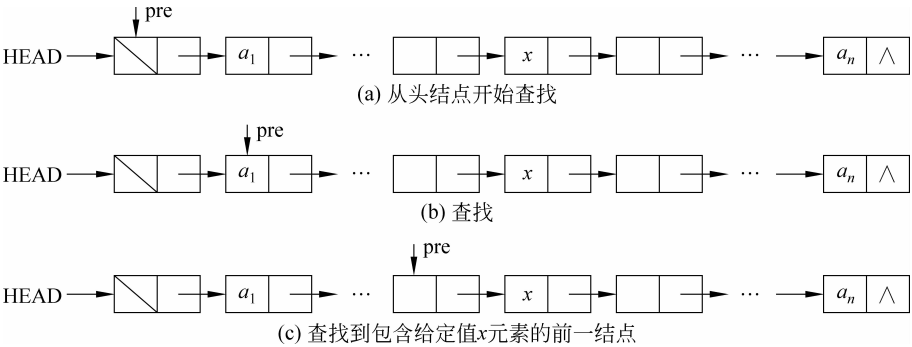


图 3-14　线性单链表的查找（按照值进行查找）

（2）按照序号进行查找：从线性单链表的头指针出发，顺着链表逐个结点往下扫描直到第 i 个结点为止。若查找到结点（$1 \leqslant i \leqslant n$），则返回结点存储位置（指针），否则返回 NULL。在实际应用中，为了便于操作，在线性单链表中查找第 i 个结点的前一结点存储位置（可用指针 pre 指向），就可以方便地在该结点后插入新的结点或者删除该结点后的结点。查找的结果如下。若查找到结点（$1 \leqslant i \leqslant n$），则返回的(pre)为第 i 个结点的前一结点位置；否则返回的(pre)为单链表的最后结点位置，如图 3-15 所示。

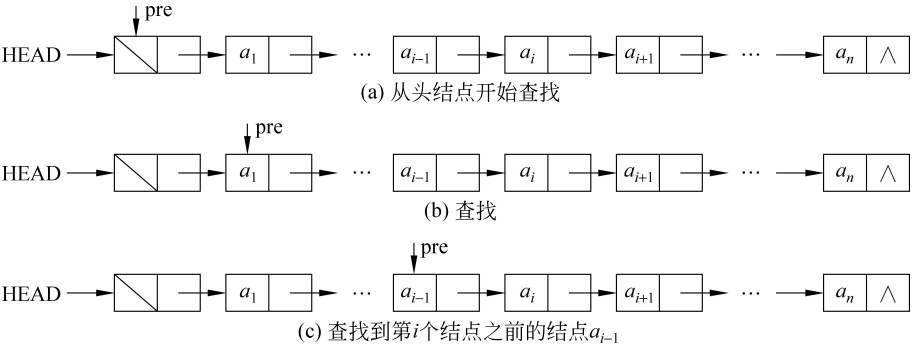


图 3-15　线性单链表的查找（按照序号进行查找）

3）线性单链表的插入

线性单链表的插入是指在线性单链表中插入一个新的数据元素。

（1）按照值进行插入是指在线性单链表中指定值数据元素之前插入一个新数据元素。运算过程如下。

① 申请新结点：生成一个新结点，输入数据 y 存放到数据域；

② 查找：在单链表中查找包含给定值 x 元素的前一结点存储位置（用指针 pre 指向）；

③ 修改链接：首先设置新结点的指针域指向结点 x，然后再设置 x 前一结点的指针域指向新结点（注意，在程序设计时此顺序不能颠倒），如图 3-16 所示。

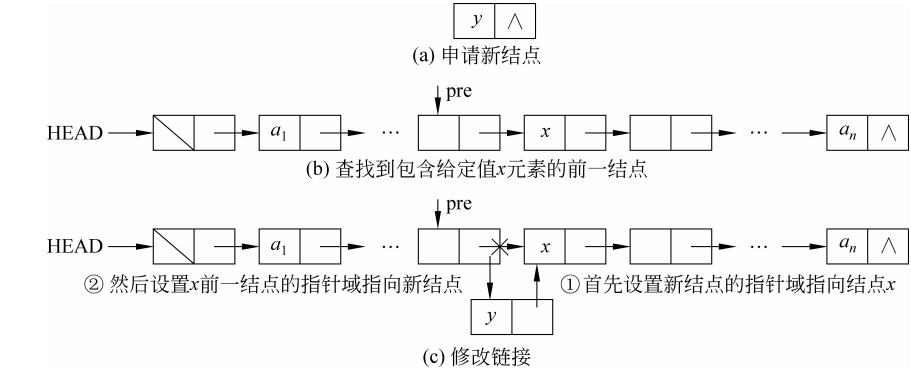


图 3-16　线性单链表的插入（按照值进行插入）

（2）按照序号进行插入是指在线性单链表中第 i 个结点之前插入一个新的结点。运算过程如下。

① 申请新结点：生成一个新结点，输入数据 x 存放到数据域；

② 查找：在单链表中查找第 i 个结点之前的结点 a_{i-1} 的存储位置（用指针 pre 指向）；

③ 修改链接：首先设置新结点的指针域指向结点 a_i，然后再设置结点 a_{i-1} 的指针域指向新结点（注意，在程序设计时此顺序不能颠倒）。从而实现三个结点 a_{i-1}，x 和 a_i 之间的逻辑关系的变化，以便将新结点插入到单链表的第 i 个结点之前的位置上，即插入到 a_{i-1} 与 a_i 之间，如图 3-17 所示。

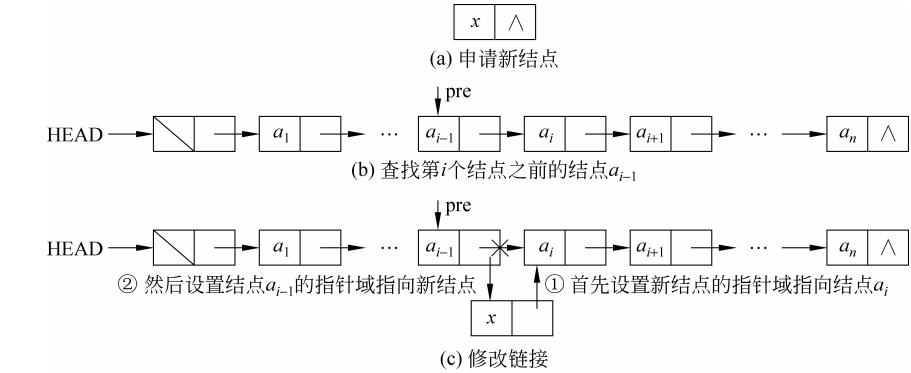


图 3-17　线性单链表的插入（按照序号进行插入）

可以看到，在线性单链表的插入过程中，不需要移动数据元素，只要改变相关结点的指针就可以实现，提高了效率。

4）线性单链表的删除

线性单链表的删除是指在线性单链表中删除一个数据元素。

（1）按照值进行删除是指在线性单链表中删除指定值的数据元素。运算过程如下。

① 查找：在单链表中查找包含给定值 x 元素的前一结点存储位置（用指针 pre 指向）；

② 修改链接：设置 x 的前一结点指向 x 的直接后继结点，即 x 前一结点的指针域改变为存放结点 x 指针域的值（为 x 直接后继结点的存储地址），这样就把结点 x 从链上摘下；

③ 释放结点：最后释放结点 x 所占用的存储空间，如图 3-18 所示。

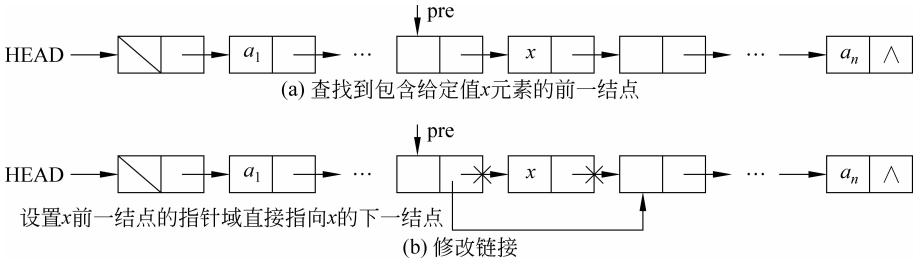


图 3-18　线性单链表的删除（按照值进行删除）

（2）按照序号进行删除是指在线性单链表中删除第 i 个结点。运算过程如下。

① 查找：在单链表中查找第 i 个结点之前的结点 a_{i-1} 的存储位置（用指针 pre 指向）；

② 修改链接：设置 a_{i-1} 指向 a_i 的直接后继结点 a_{i+1}，即把 a_{i-1} 的指针域改变为存放结点 a_i 指针域的值（为 a_{i+1} 的存储地址），就能把结点 a_i 从链上摘下；

③ 释放结点：最后释放结点 a_i 所占用的存储空间，如图 3-19 所示。

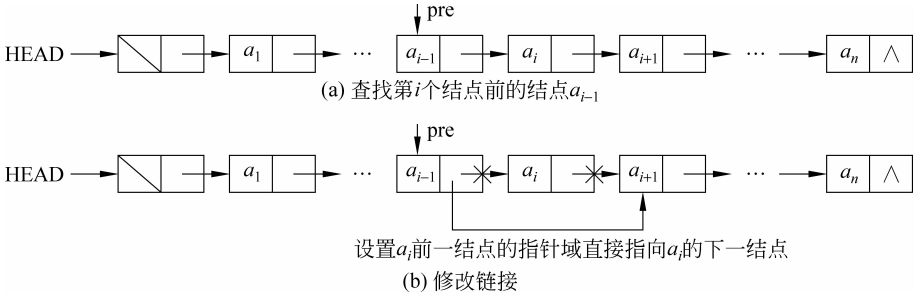


图 3-19　线性单链表的删除（按照序号进行删除）

可以看到，在线性单链表的删除过程中，不需要移动数据元素，只要改变被删除结点的前一结点指针域，使其指向被删除结点的直接后继结点就可以实现，提高了效率。

链表的优缺点如下。

（1）优点：在链表中进行插入、删除操作时只需要修改指针而不需要移动表中的数据元素，对于频繁进行插入、删除操作的线性表适合采用链表做存储结构。

(2) 缺点：对链表中的数据元素只能顺着链表进行顺序存取而不能进行随机存取。

2. 双链表

在线性单链表中，每个结点只包含一个指向其直接后继的指针(地址)，因此从某一个结点出发，由这个指针可以很方便地找到后继结点，但是却不能找到前驱结点。为了能找到前驱结点，必须从头指针重新开始另一次查找。

为了快速地找到任意一个结点的前驱结点和后继结点，在线性单链表的每个结点中再增加一个指针域，用来指向该结点的直接前驱结点。每个结点包含两个指针(如图 3-20 所示)，其中指向直接前驱结点的指针称为左指针(llink)，指向直接后继结点的指针称为右指针(rlink)，由此形成的线性链表就有两个不同方向的链，则称为双向链表，简称为双链表(Double Linked List)，如图 3-21 所示。

图 3-20　双链表的结点形式

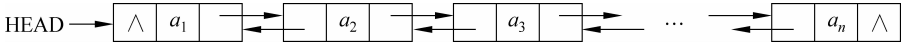


图 3-21　双向链表(不带头结点)

其中，data 是数据域，用来存放结点的值；llink 是指向直接前驱结点的左指针，一般用来存放结点的直接前驱的地址(或位置)；rlink 是指向直接后继结点的右指针，一般用来存放结点的直接后继的地址(或位置)。

为了操作方便，可以在双链表第一个结点之前再增加一个结点，称为头结点。头结点的数据域为任意或依据需要设置，头结点的指针域存储第一个结点的存储地址(即指向第一个结点)。此时头指针 HEAD 指向头结点，而头结点指向双向链表的第一个结点，如图 3-22 所示。

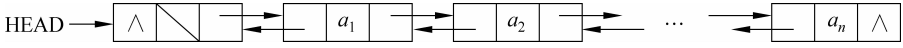


图 3-22　双向链表(带头结点)

3. 循环链表

虽然线性单链表的插入、删除运算很方便，但是在运算过程中必须单独考虑空表和第一个结点等特殊情况，使得空表和非空表的运算不统一而需要采用不同的方法；另外，若要查找某一结点的前驱结点，必须从头指针重新开始另一次查找。为了避免这些缺点，可以使用线性表的另一种链式存储结构——循环链表。

将单链表中最后一个结点的指针域由空(NULL)改变为指向头结点而使链表头尾相接形成环状，称为单向循环链表，简称为循环链表(Circular Linked List)。

在单向循环链表中，只是对表的链接方式稍微改变一些，就使得可以从任意一个结点

出发来访问表中所有其他的结点，提高了查找的效率。而线性单链表却做不到这一点。

在程序设计时，为了操作方便而使空表和非空表的处理保持一致，可在循环链表中再增加一个头结点，它的数据域可以任意或依据需要设置，指针域指向链表第一个结点，如图 3-23 所示。

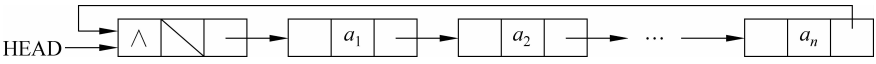


图 3-23　单向循环链表(带头结点)

循环链表具有如下两个特点。

(1) 循环链表增加一个头结点，其数据域可以依据需要设置，其指针域指向第一个结点。即循环链表的头指针 HEAD 指向头结点，而头结点指向第一个结点。

(2) 循环链表最后一个结点的指针域指向头结点而不是空。

由于增加了一个表头结点，所以在任何情况下，循环链表至少有一个结点存在，在对循环链表进行插入、删除运算的过程中，就实现了空表和非空表的运算统一。

在实际应用中，循环链表的插入和删除运算与线性单链表的插入和删除运算方法基本相同。但是由于在循环链表的插入、删除运算过程中实现了空表和非空表的运算统一，所以使得利用高级语言(C 语言)具体实现循环链表的查找、插入和删除运算比实现线性单链表的查找、插入和删除运算更加简单方便。

链式存储结构的存储方式既可以用来表示线性结构，也可以用来表示非线性结构。

3.3　栈 和 队 列

3.3.1　栈及其基本运算

1. 栈的基本概念

栈(Stack)是一种特殊的线性表，它是限定仅在一端进行插入或删除操作的线性表，即栈是一种运算受限的线性表。其中，允许进行插入或删除操作的一端称为栈顶(top)，不允许进行插入、删除操作的另一端称为栈底(bottom)。不含数据元素的栈称为空栈。

栈根据“先进后出”或“后进先出”的原则组织数据，栈有时也被称为“先进后出的线性表”或“后进先出的线性表”。一般用指针 top 指向栈顶位置，用指针 bottom 指向栈底位置。向栈中插入一个数据元素称为入栈，从栈中删除一个数据元素称为退栈。

由于栈也是一种线性表，因此也可以采用线性表的顺序存储和链式存储这两种存储结构。以顺序存储结构存储的栈称为顺序栈。以链式存储结构存储的栈称为带链的栈。

下面举例说明栈结构的特点。假设有一个火车维修处，其中一侧为封闭的，现有 6 辆火车，编号 1、2、3、4、5、6，按照编号的顺序进入维修处。此时，若编号 1 的火车要退出，必须要等编号 6、5、4、3、2 的火车依次退出后才行。这个火车维修处可以看作一个栈，编号 6 火车的位置为栈顶，编号 1 火车的位置为栈底，火车的进入和退出可以看作栈的插入和

删除，并且都在栈顶进行，如图 3-24 所示。

2. 栈的顺序存储及其基本运算

与一般的线性表类似，在程序设计时，可以使用一维数组作为栈的顺序存储空间，即可以利用一组地址连续的存储单元依次存放从栈底到栈顶的所有数据元素。用顺序存储结构存储的栈称为顺序栈。可以设置 n 表示栈的最大存储空间，并假设栈顶指针为 top，如图 3-25 所示。

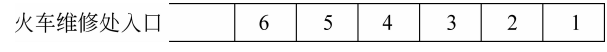


图 3-24　火车维修处的示意图

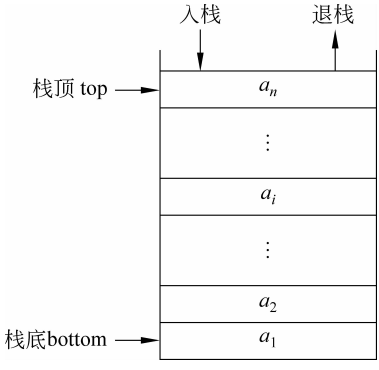


图 3-25　栈的示意图

栈的顺序存储结构下的基本运算包括入栈运算、退栈运算、读栈运算。

1）顺序栈的入栈运算

顺序栈的入栈运算是指在栈顶位置插入一个新的数据元素。运算过程如下。

（1）修改指针：将栈顶指针加 1(top 加 1)；

（2）插入：在当前栈顶指针所指向的位置将新的数据元素插入。

在顺序栈的入栈运算过程中，若栈顶指针已经指向栈存储空间的最后位置(即 top＝n)，表明此时栈的空间已满，不能再进行入栈运算，否则会产生栈的“上溢”错误。

2）顺序栈的退栈运算

顺序栈的退栈运算是指在栈顶位置取走一个数据元素并且把它赋给某个变量。运算过程如下。

（1）退栈：将栈顶指针所指向的栈顶元素读取后并且赋给一个变量；

（2）修改指针：将栈顶指针减 1(top 减 1)。

在顺序栈的退栈运算过程中，若栈顶指针已经为 0 时(即 top＝0)，表明此时栈空，不能再进行退栈运算，否则会产生栈的“下溢”错误，如图 3-26 所示。

3）顺序栈的读栈运算

顺序栈的读栈运算是指在栈顶位置读取一个数据元素并且把它赋给某个变量。运算过程为将栈顶指针所指向的栈顶元素读取并赋给一个变量，栈顶指针保持不变。

在顺序栈的读栈运算过程中，若栈顶指针已经为 0 时(即 top＝0)，表明此时栈空，不能再进行读栈运算，即读不到栈顶元素。

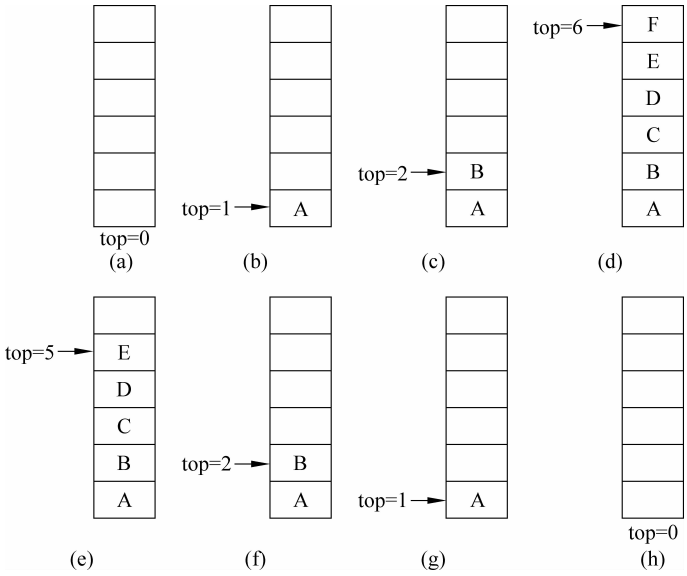


图 3-26　顺序栈的入栈和退栈

3. 栈的链式存储及其基本运算

与一般的线性表类似，在程序设计时，栈也可以使用链式存储结构。用链式存储结构存储的栈称为带链的栈，简称为链栈。它其实就是运算受限的单链表，其插入和删除操作仅限制在链表的表头位置上进行，所以，栈顶指针就是链表的头指针，头指针指向栈顶结点（不带头结点）或者头结点（带头结点），如图 3-27 和图 3-28 所示。

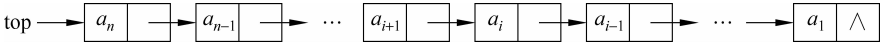


图 3-27　带链的栈（不带头结点）

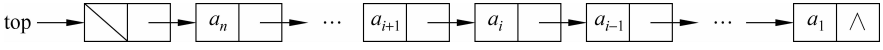


图 3-28　带链的栈（带头结点）

栈的链式存储结构下的基本运算包括带链栈的入栈运算、带链栈的退栈运算。

1）带链栈的入栈运算

带链栈的入栈运算是指在栈顶位置插入一个新的数据元素，如图 3-29 所示。运算过程如下。

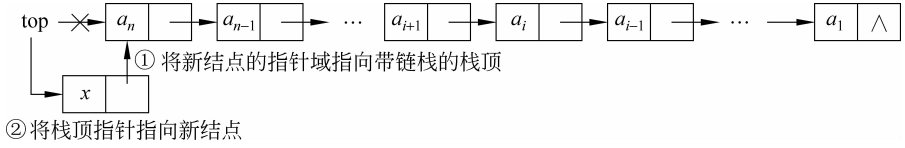


图 3-29　带链栈的入栈运算（不带头结点）