第 2 章　基本数据结构与算法

教学目标

- 掌握算法的基本概念。
- 掌握基本数据结构及其操作。
- 掌握基本排序和查找算法。

教学重、难点

- 算法的基本概念；算法复杂度的概念和意义（时间复杂度与空间复杂度）。
- 数据结构的定义；数据的逻辑结构与存储结构；数据结构的图形表示；线性结构与非线性结构的概念。
- 线性表的定义；线性表的顺序存储结构及其插入与删除运算。
- 栈和队列的定义；栈和队列的顺序存储结构及其基本运算。
- 线性单链表、双向链表与循环链表的结构及其基本运算。
- 树的基本概念；二叉树的定义及其存储结构；二叉树的前序、中序和后序遍历。
- 顺序查找与二分法查找算法；基本排序算法（交换类排序、选择类排序、插入类排序）。

2.1　算法

当我们要用计算机来编写程序的时候，总要首先想好这个程序是做什么的？应该如何实现程序的目标？应该先进行什么处理，后进行什么处理？所处理的数据格式是什么？遇到一些复杂的问题，我们可能还需要考虑采用什么数学方法来处理。这一切都涉及一个专业名词——“算法”。

2.1.1　算法的基本概念

所谓算法，是指对解题方案的准确而完整的描述。

算法（Algorithm）是一系列解决问题的清晰指令，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，能够对一定规范的输入，在有限时间内获得所要求的输出。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。算法不等于程序，也不等于计算方法。当然，程序也可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序的编制不可能优于算法的设计。算法可以使用自然语言、伪代码、流程图等多种不同的方法来描述。

1. 算法的基本特征

算法实际上是解决问题的方法。一个算法应该具有以下五个重要的特征：

（1）有穷性（Finiteness）。算法的有穷性是指算法必须能在执行有限个步骤之后终止。

（2）确定性（Definiteness）。算法的每一步骤必须有确切的定义，读者理解不会产生二义性。并且在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。

（3）可行性（Effectiveness）。算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的，即每个操作都可以在有限时间内完成。

（4）输入（Input）。一个算法有 0 个或多个输入，以描述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（Output）。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。

2. 算法的描述

一个算法可以用自然语言、计算机程序语言或其他语言来说明，唯一的要求是该说明必须精确地描述计算过程。例如，一个需在计算机上运行的程序（程序也是算法）必须严格按照语法规定用机器语言或汇编语言或高级语言编写，而一个为了人的阅读和交流的算法可以用介于自然语言和程序语言之间的伪语言或框图等其他形式来描述。算法的控制结构往往类似于 Pascal、C、Java 等程序语言，但其中可使用任何表达能力强的方法使算法表达更加清晰和简洁，而不至于陷入具体的程序语言的某些细节。

3. 算法的基本要素

算法一般由两个基本要素构成：一个是对数据对象的运算和操作，一个是算法的控制结构。

算法中对数据的运算和操作指的是每个算法实际上是按照解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。通常基本的运算操作有四类：算术运算、逻辑运算、关系运算和数据传输。算法的控制结构指的是算法中各操作之间的执行顺序，算法的功能不仅取决于所选用的操作，还与各操作之间的执行顺序有关。基本的控制结构包括顺序结构、选择结构和循环结构。

4. 算法的设计要求

通常设计一个好的算法应考虑达到以下目标：

（1）正确性。一个算法应当能够解决具体问题。其“正确性”可分为以下几个方面：①不含逻辑错误；②对于几组输入数据能够得出满足要求的结果；③对于精心选择的典型、苛刻的输入数据都能得到要求的结果；④对于一切合法的输入都能输出满足要求的结果，通常以第③层意义的正确作为衡量一个程序是否合格的标准。

（2）可读性。算法应该可以用能够被人理解的形式表示。算法设计的目的就是让大家相互交流，太复杂的、不能被程序员所理解的算法难以在程序设计中采用。

（3）健壮性。当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫明其妙的输出结果。

（4）效率与低存储量的需求。高效率和低存储量是优秀程序员追求的目标。效率指的是算法执行时间，对于一个问题如果有多个算法可以解决，执行时间短的算法效率高。存储量需求指算法执行进程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规模有关。占用存储量最小、运算时间最少的算法就是最好的算法。但是在实际中，运行时间和存储空间往往是一对矛盾，要根据具体情况选择更优先考虑哪一个因素。

5. 算法设计的基本方法

（1）列举法。

列举法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从中找出那些符合要求的候选解作为问题的解。

（2）归纳法。

归纳法的基本思想是通过列举少量的特殊情况，经过分析，最后找出一般的关系。显然，归纳法要比列举法更能反映问题的本质，并且可以解决列举量为无限的问题。但是，从一个实际问题中总结归纳出一般的关系，并不是一件容易的事情，尤其是要归纳出一个数学模型更为困难。从本质上讲，归纳就是通过观察一些简单而特殊的情况，最后总结出有用的结论或解决问题的有效途径。

（3）递归法。

人们在解决一些复杂问题时，为了降低问题的复杂程度（如问题的规模等），一般总是将问题逐层分解，最后归结为一些最简单的问题。这种将问题逐层分解的过程，实际上并没有对问题进行求解，而只是当解决了最后那些最简单的问题后，再沿着原来分解的逆过程逐步进行综合，这就是递归的基本思想。由此可以看出，递归的基础也是归纳。在工程实际中，有许多问题就是用递归来定义的，数学中的许多函数也是用递归来定义的。递归在可计算性理论和算法设计中占有很重要的地位。递归分为直接递归与间接递归两种。如果一个算法 P 显式地调用自己则称为直接递归；如果算法 P 调用另一个算法 Q，而算法 Q 又调用算法 P，则称为间接递归。

（4）递推法。

递推法是利用问题本身所具有的一种递推关系求问题解的一种方法。它把问题分成若干步，找出相邻几步的关系，从而达到目的，此方法称为递推法。

（5）回溯法。

前面讨论的递推和递归算法，其本质上是对实际问题进行归纳的结果，而减半递推技术也是归纳法的一个分支。在工程上，有些实际问题很难归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限的列举。对于这类问题，一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索，然后沿着这个线索逐步试探。对于每一步的试探，若试探成功，就得到问题的解；若试探失败，就逐步回退，换别的路线再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

2.1.2　算法复杂度

算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。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。算法分析的目的在于改进算法和选择合适算法。一个算法的评价主要从时间复杂度和空间复杂度来考虑。

1. 时间复杂度

（1）时间频度。

一个算法执行所耗费的时间，从理论上是不能算出来的，必须上机运行测试才能知道。但我们不可能也没有必要对每个算法都上机测试，只需知道哪个算法花费的时间多，哪个算法花费的时间少就可以了。并且一个算法花费的时间与算法中语句的执行次数成正比，哪个算法

中语句执行次数多，它花费时间就多。一个算法中的语句执行次数称为语句频度或时间频度。记为 T(n)。

（2）时间复杂度。

在刚才提到的时间频度中，n 称为问题的规模，当 n 不断变化时，时间频度 T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度概念。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，用 T(n)表示，若有某个辅助函数 f(n)，使得当 n 趋近于无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称 f(n)是 T(n)的同数量级函数。记作 T(n)=O(f(n))，称 O(f(n)) 为算法的渐进时间复杂度，简称时间复杂度。

在各种不同算法中，若算法中语句执行次数为一个常数，则时间复杂度为 O(1)，另外，在时间频度不相同时，时间复杂度有可能相同，如 $T(n)=n^2+3n+4$ 与 $T(n)=4n^2+2n+1$ 它们的频度不同，但时间复杂度相同，都为 $O(n^2)$。

按数量级递增排列，常见的时间复杂度有：

常数阶 O(1)，对数阶 $O(\log_2 n)$（以 2 为底 n 的对数，下同），线性阶 O(n)，线性对数阶 $O(\log_2 n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，k 次方阶 $O(n^k)$，指数阶 $O(2^n)$。随着问题规模 n 的不断增大，上述时间复杂度不断增大，算法的执行效率越低。

2. 空间复杂度

与时间复杂度类似，空间复杂度是指算法在计算机内执行时所需存储空间的度量。记作：

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

其中 n 为问题的规模（或大小），我们一般所讨论的是除正常占用内存开销外的辅助存储单元规模。讨论方法与时间复杂度类似，不再赘述。

2.2 数据结构的基本概念

数据结构作为计算机的一门学科，主要研究和讨论以下 3 个方面的问题：

（1）数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构。

（2）在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构。

（3）对各种数据结构进行的运算。

实际问题中的各数据元素之间总是相互关联的。所谓数据处理，是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。在数据处理领域中，建立数学模型有时并不十分重要，事实上，许多实际问题是无法表示成数学模型的。人们最感兴趣的是知道数据集合中各数据元素之间存在什么关系，为了提高处理效率，应如何组织它们，即如何表示所需要处理的数据元素。

2.2.1 数据结构的定义

在了解数据结构的定义之前我们先了解一些基本概念。

数据（data）：数据是信息的载体。它能够被计算机识别、存储和加工处理，是计算机程序加工的“原料”。随着计算机应用领域的扩大，数据的范畴包括：数值、字符串、图像和声音等各种多媒体信息。

数据元素（data element）：数据元素是数据的基本单位。数据元素也称元素、结点、顶点、记录。如一年四季的季节名可以作为季节的数据元素，表示家庭成员的各成员名（父亲、母亲、儿子、女儿）可以作为家庭成员的数据元素。

数据项：一个数据元素可以由若干个数据项（也可称为字段、域、属性）组成。数据项是具有独立含义的最小标识单位。

数据结构：数据结构是指相互有关联的数据元素的集合，即数据在计算机中的组织形式。数据结构一般包括以下三方面内容：数据的逻辑结构、数据的存储结构和数据的运算。

2.2.2　数据的逻辑结构

数据的逻辑结构是指反映数据元素之间逻辑关系的数据结构。它包括两个要素：一是数据元素集合，通常记为 D，二是数据元素之间的关系，它反映了数据元素集合中数据元素之间的关系，通常记为 R，于是，一个数据结构可表示为 B=（D，R），其中 B 表示数据结构。

2.2.3　数据的存储结构

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构（也称数据的物理结构）。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。数据结构的存储方式有以下四种：

（1）顺序存储方法：该方法把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为顺序存储结构，通常借助程序语言的数组描述。该方法主要应用于线性的数据结构。非线性的数据结构也可通过某种线性化的方法实现顺序存储。

（2）链式存储方法：该方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系由附加的指针字段表示。由此得到的存储表示称为链式存储结构，通常借助于程序语言的指针类型描述。

（3）索引存储方法：该方法通常在存储结点信息的同时，还建立附加的索引表。索引表由若干索引项组成。若每个结点在索引表中都有一个索引项，则该索引表称为稠密索引。若一组结点在索引表中只对应一个索引项，则该索引表称为稀疏索引。索引项的一般形式是：（关键字、地址）。关键字是能唯一标识一个结点的那些数据项。稠密索引中索引项的地址指示结点所在的存储位置，稀疏索引中索引项的地址指示一组结点的起始存储位置。索引存储方法类似于我们熟悉的书的目录。

（4）散列存储方法：该方法的基本思想是：根据结点的关键字直接计算出该结点的存储地址。

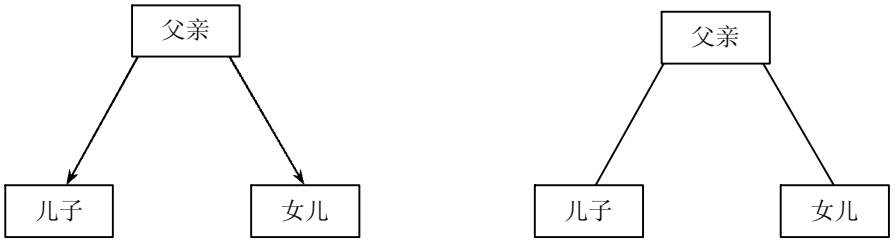
2.2.4　数据的运算

数据的运算即对数据施加的操作。数据的运算定义在数据的逻辑结构上，每种逻辑结构都有一个运算的集合。最常用的检索、插入、删除、更新、排序等运算实际上只是在抽象的数据上所施加的一系列抽象的操作。所谓抽象的操作；是指我们只知道这些操作是“做什么”，而无须考虑“如何做”。只有确定了存储结构之后，才考虑如何具体实现这些运算。

2.2.5 数据结构的图形表示

一个数据结构除了用二元关系表示外，还可以直观地用图形表示。在数据结构的图形表示中，对于数据集合 D 中的每一个数据元素用中间标有元素值的方框表示，一般称为数据结点，并简称为结点。为了进一步表示各数据元素之间的前后件关系，对于关系 R 中的每一个二元组，用一条有向线段从前件结点指向后件结点。例如，反映家庭成员间辈分关系的数据结构可以用如图 2-1（a）所示的图形表示。

显然，用图形方式表示一个数据结构是很方便的，并且也比较直观。有时在不会引起误会的情况下，在前件结点到后件结点连线上的箭头可以省去。例如，在图 2-1（a）中，即使将“父亲”结点与“儿子”结点连线上的箭头以及“父亲”结点与“女儿”结点连线上的箭头都去掉，如图 2-1（b）所示，同样表示了“父亲”是“儿子”与“女儿”的前件，“儿子”与“女儿”均是“父亲”的后件，而不会引起误会。



（a）家庭成员间辈分关系数据结构的图形表示　　（b）家庭成员间辈分关系数据结构的图形表示

图 2-1　家庭成员间辈分关系数据结构

2.2.6 线性结构与非线性结构

如果在一个数据结构中一个数据元素都没有，则称该数据结构为空的数据结构。在一个空的数据结构中插入一个新的元素后就变为非空数据结构；在只有一个数据元素的数据结构中，将该元素删除后就变为空的数据结构。根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

如果一个非空的数据结构满足两个条件：一是有且只有一个根结点，二是每一个结点最多有一个直接前驱，也最多有一个直接后继。则称该数据结构为线性结构，又称线性表。需要注意的是，在一个线性结构中插入或删除任何一个结点后还应该是线性结构，否则，不能称为线性结构。如果一个数据结构不满足上述两个条件之一，则称为非线性结构。显然，在非线性结构中，各数据元素之间的前后关系要比线性结构复杂。因此，对非线性结构的存储与处理比线性结构复杂得多。线性结构与非线性结构都可以是空的数据结构。一个空的数据结构究竟是属于线性结构还是属于非线性结构，这要根据具体情况来确定。如果对该数据结构的运算是按线性结构的规则来处理的，则属于线性结构；否则属于非线性结构。

2.3　线性表及其顺序存储结构

2.3.1　线性表的基本概念

线性表是由 n(n≥0)个数据元素 $a_1,a_2,a_3,a_4 \ldots a_n$ 组成的一个有限序列，表中的每一个数据元素，除了第一个外，有且只有一个直接前趋，除了最后一个外，有且只有一个直接后继，即线性表或是一个空表，或可以表示为$(a_1,a_2,a_3,a_4 \ldots a_n)$，其中 a_i 通常称其为线性表中的一个结点。显然，线性表是一种线性结构。数据元素在线性表中的位置只取决于它们自己的序号，即数据元素之间的相对位置是线性的。

非空线性表有如下一些结构特征：

（1）有且只有一个根结点 a_1，它无直接前趋。

（2）有且只有一个终端结点 a_n，它无直接后继。

（3）除根结点与终端结点外，其他所有结点有且只有一个直接前趋，也有且只有一个直接后继。线性表中结点的个数 n 称为线性表的长度。当 n=0 时，称为空表。

2.3.2　线性表的顺序存储结构

在计算机中存放线性表的一种最简单的方法是顺序存储，也称为顺序分配。

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称为顺序表。线性表的顺序存储结构具有以下两个基本特点：

（1）线性表中所有元素所占的存储空间是连续的。

（2）线性表中各元素在存储空间中是按逻辑顺序依次存放的。由此可知，在线性表的顺序存储结构中，其前后件两个元素在存储空间中是紧邻的，且前件元素一定存储在后件元素的前面。

线性表中结点 a_i 的存储地址，设线性表中所有结点的类型相同，则每个结点所占用存储空间大小亦相同。假设表中每个结点占用 f 个存储单元，其中第一个单元的存储地址则是该线性表的存储地址，并设表中开始结点。开始结点的存储地址（简称为基地址）是 $LOC(a_1)$，那么结点 a_i 的存储地址 $LOC(a_i)$可通过下式计算：

$LOC(a_i)=LOC(a_1)+(i-1)*f \quad 1 \leqslant i \leqslant n$

在顺序表中，每个结点 a_i 的存储地址是该结点在表中的位置 i 的线性函数。只要知道基地址和每个结点的大小，就可在相同时间内求出任一结点的存储地址。因此，线性表是一种随机存取结构。

在计算机中顺序存储结构如图 2-2 所示。

2.3.3　线性表的插入

线性表的插入运算是指在表的第 i(1≤i≤n+1)个位置上，插入一个新结点 J，使长度为 n 的线性表$(a_1,a_2,\ldots,a_n)$变成长度为 n+l 的线性表$(a_1,a_2,\cdots,a_{n+1})$。由于向量空间大小在声明时确定，当 L→length≥ListSize 时，表示空间已满，不可再做插入操作。当插入位置 i 的值为 i>n 或 i<l 时为非法位置，不可做正常插入操作。顺序表插入操作过程：在顺序表中，结点的物理顺序必

须和结点的逻辑顺序保持一致，因此必须将表中位置为 n,n-l,…,i 上的结点，依次后移到位置为 n+1,n,…,i+l 的结点上，空出第 i 个位置，然后在该位置上插入新结点 J，并将表长加 l。仅当插入位置 i=n+l 时，才无须移动结点，直接将结点插入表的末尾即可。

存储位置	下标	
		⋮
LOC(a_1)	0	a_1
LOC(a_1+c)	1	a_2
		⋮
LOC(a_1)+(i−1)*c	i	a_i
		⋮
LOC(a_1)+(n−1)*c	n	a_n

图 2-2　线性表的顺序存储结构

2.3.4　线性表的删除

一般来说，设长度为 n 的线性表为

$$(a_1, a_2, \cdots, a_i, \cdots, a_n)$$

现要删除第 *i* 个元素，删除后得到长度为 n−1 的线性表为

$$(a'_1, a'_2, \cdots, a'_j, \cdots, a'_{n-1})$$

则删除前后的两线性表中的元素满足如下关系：

$$a'_j = \begin{cases} a_j, & 1 \leqslant j \leqslant i-1 \\ a_{j+1}, & i \leqslant j \leqslant n-1 \end{cases}$$

在一般情况下，要删除第 i(1≤i≤n)个元素时，则要从第 $i+1$ 个元素开始，直到第 n 个元素之间共 $n-i$ 个元素依次向前移动一个位置。删除结束后，线性表的长度就减少了 1。

显然，在线性表采用顺序存储结构时，如果删除运算在线性表的末尾进行，即删除第 n 个元素，则不需要移动表中的元素；但如果要删除线性表中的第 1 个元素，则需要移动表中所有的元素。在一般情况下，如果要删除第 j(1≤j≤n)个元素，则原来第 j 个元素之后的所有元素都必须依次往前移动一个位置。在平均情况下，要在线性表中删除一个元素，需要移动表中一半的元素。因此，在线性表顺序存储的情况下，要删除一个元素，其效率也是很低的，特别是在线性表比较大的情况下更为突出，由于数据元素的移动而消耗较多的处理时间。

由线性表在顺序存储结构下的插入与删除运算可以看出，线性表的顺序存储结构对于小线性表或者其中元素不常变动的线性表来说是合适的，因为顺序存储的结构比较简单。但这种顺序存储的方式对于元素经常需要变动的大线性表就不太合适了，因为插入与删除的效率比较低。

2.4　栈与队列

栈和队列是两种特殊的线性表，它们的逻辑结构和线性表相同，只是其运算规则较线性表有更多的限制，故又称它们为运算受限的线性表。栈和队列被广泛应用于各种程序设计中。

2.4.1　栈及其基本运算

1．栈的基本概念

栈（Stack）是一种只允许在表的一端进行插入和删除运算的线性表，它是一种操作受限的线性表。在这种特殊的线性表中，其插入与删除运算都只在线性表的一端进行，即在这种线性表的结构中，一端是封闭的，不允许进行插入与删除元素；另一端是开口的，允许插入与删除元素。在栈中，允许插入与删除的一端称为栈顶（top），而不允许插入与删除的另一端称为栈底（bottom）。栈顶元素总是最后被插入的元素，从而也是最先能被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素，即栈是按照“先进后出”（FILO）或“后进先出”（LIFO）的原则组织数据的，因此，栈也称为“先进后出”表或“后进先出”表。由此可以看出，栈具有记忆作用。通常用指针 top 来指示栈顶的位置，用指针 bottom 指向栈底。往栈中插入一个元素称为入栈运算，从栈中删除一个元素（即删除栈顶元素）称为退栈运算。栈顶指针 top 动态反映了栈中元素的变化情况。图 2-3 是栈的示意图。

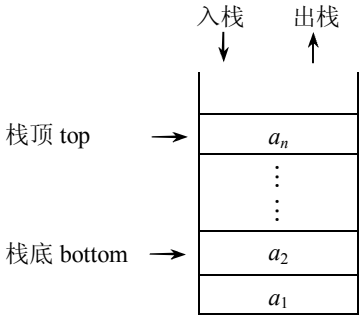


图 2-3　栈示意图

2．栈的顺序存储及其运算

栈的顺序存储结构简称为顺序栈，它是运算受限的顺序表。

图 2-4 示意顺序栈中元素用向量存放，栈底位置是固定不变的，可设置为向量两端的任意一个端点，栈顶位置是随着进栈和退栈操作而变化的，用一个整型量 top（通常称 top 为栈顶指针）来指示当前栈顶位置。top=0 表示栈空，top=StackSize-1 表示栈满。上溢：当栈满时，再做进栈运算产生空间溢出的现象。上溢是一种出错状态，应设法避免。下溢：当栈空时，做退栈运算产生的溢出现象。下溢是正常现象，常用作程序控制转移的条件。

顺序栈的基本运算。设 S 是 Stack 类型的指针变量。若栈底位置在向量的低端，即 S→data[0] 是栈底元素。

（1）进栈操作。

进栈操作是指在栈顶位置插入一个新元素。进栈时，需要将 S→top 加 l，将入栈元素加入

到新的栈顶下标所指的位置上。

（2）退栈操作。

退栈操作是指取出栈顶元素。退栈时，需将 S→top 减 l。

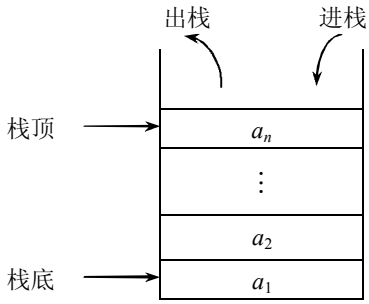


图 2-4　栈示意图

2.4.2　队列及其基本运算

队列（queue）是指允许在一端进行插入、而在另一端进行删除的线性表。允许插入的一端称为队尾，通常用一个称为尾指针（rear）的指针指向队尾元素，即尾指针总是指向最后被插入的元素；允许删除的一端称为排头（也称为队头），通常用一个排头指针（front）指向排头元素的前一个位置。显然，在队列这种数据结构中，最先插入的元素将最先能够被删除，反之，最后插入的元素将最后才能被删除。因此，队列又称为“先进先出”（First In First Out，FIFO）或“后进后出”（Last In Last Out，LILO）的线性表，它体现了“先来先服务”的原则。在队列中，队尾指针 rear 与排头指针 front 共同反映了队列中元素动态变化的情况。

1．队列的基本概念

队列是一种特殊的线性表，它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

在队列这种数据结构中，最先插入的元素将是最先被删除的元素；反之最后插入的元素将是最后被删除的元素，因此队列又称为“先进先出”（FIFO）的线性表，如图 2-5 所示。

队列空的条件：front=rear

队列满的条件：rear = MAXSIZE

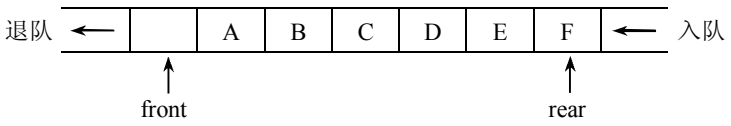


图 2-5　具有 6 个元素的队列示意图

2．循环队列及其运算

在实际应用中，队列的顺序存储结构通常采用循环队列的形式。

所谓循环队列，是指将队列存储空间的最后一个位置绕到第一个位置，形成逻辑上的形状空间，供队列循环使用，如图 2-6 所示。

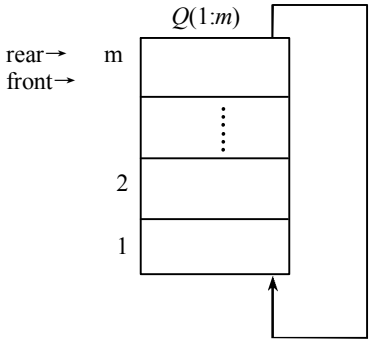


图 2-6　循环队列存储空间示意图

在循环队列中，用队尾指针（real）指向队列的队尾元素，用排头指针（front）指向排头的前一个位置，因此，从排头指针指向的后一个位置直到队尾指针指向的位置之间所有的元素均为队列中的元素。

循环队列的初始状态为空，即 real=front。

2.5　线性链表

2.5.1　线性链表的基本概念

用链接方式存储的线性表简称为链表（linked list）。链式存储是最常用的存储方式之一，它不仅可用来表示线性表，而且可用来表示各种非线性的数据结构。链表的具体存储表示为：

（1）用一组任意的存储单元来存放线性表的结点（这组存储单元既可以是连续的，也可以是不连续的）。

（2）链表中结点的逻辑次序和物理次序不一定相同。为了能正确表示结点间的逻辑关系，在存储每个结点值的同时，还必须存储指示其后继结点的地址（或位置）信息，称为指针（pointer）或链（link）。

链表的结点结构如图 2-7 所示，其中 data 域是存放结点值的数据域，next 域是存放结点的直接后继的地址（位置）的指针域（链域）。

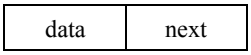


图 2-7　链表的结点结构

链表是通过每个结点的链域将结点结构线性表的 n 个结点按其逻辑顺序链接在一起的。每个结点只有一个链域的链表称为单链表（single linked list）。

头指针 head 和终端结点指针域的表示。单链表中每个结点的存储地址是存放在其前趋结点 next 域中，开始结点无前趋，头指针 head 指向开始结点。链表由头指针唯一确定，单链表可以用头指针的名字来命名。

单链表的一般图示法。由于我们只注重结点间的逻辑顺序，并不关心每个结点的实际位置，可以用箭头来表示链域中的指针，线性表（$a_1,a_2,\ldots,a_n$）的单链表就可以表示为图 2-8 的形式。