第3章 线 性 表

学习目标

本章主要介绍线性表的定义、相应的抽象数据类型和接口类，以及分别利用顺序和链接两种存储结构实现的相应线性表类，并且讨论了多项式和稀疏矩阵的线性表表示和计算。通过本章的学习，要求达到：

- 理解线性表的定义、相应的抽象数据类型和接口类的定义；
- 掌握在利用顺序存储结构实现的线性表类中各种运算的具体方法和算法描述；
- 掌握在利用链接存储结构实现线性表类中各种运算的具体方法和算法描述；
- 掌握顺序存储和链接存储的有序表的继承性以及特有运算的定义和算法描述；
- 掌握多项式和稀疏矩阵的线性表表示以及相应运算的算法描述；
- 能够根据对线性表的具体应用的需要，编写出利用线性表类解决实际问题的算法。

3.1 线性表的定义和运算

3.1.1 线性表的定义

线性表是具有相同类型数据的一个有限序列，数据与数据之间在逻辑上存在着线性关系。线性表中元素的个数称为线性表的长度。线性表长度可以为0，表明是一个空表，它不含有任何元素。若线性表为一个非空表，则一般表示为：

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

即用一对圆括号把构成线性表的所有数据元素括起来。线性表中的第一个元素 a_1 称为表头元素，a_n 称为表尾元素，线性表长度 $n \geqslant 0$。

若需要给一个线性表命名，则相应的标识符通常采用大写。如可把上面的线性表用A表示，即 $A=(a_1, a_2, \cdots, a_i, a_{i+1}, \cdots, a_n)$。

线性表中的数据元素在逻辑上是按照先后位置关系线性有序排列的，即第 i 个元素 a_i 是第 i+1 个元素 a_{i+1} 的前驱，而反过来，第 i+1 个元素 a_{i+1} 又是第 i 个元素 a_i 的后继 $(1 \leqslant i < n)$。线性表中的第一个元素只有后继没有前驱，而最后一个元素只有前驱没有后继，其余每个元素既有前驱也有后继。

线性表中的元素是线性有序的，用二元组表示为：

$$\text{linear\_list}=(A,R)$$

其中：

$$A=\{a_i \mid 1\leqslant i\leqslant n, n\geqslant 0\}$$

$$R=\{<a_i, a_{i+1}> \mid 1\leqslant i\leqslant n-1\}$$

线性表是线性数据结构的一种表示方式。如对于第 1 章中讨论的线性数据结构 linearity 可用线性表表示为：

(05,01,03,08,02,07,04,06,09,10)

因此，本章对线性表的讨论就代表了对任何线性数据结构的讨论。

在日常生活中所见到的各种各样的表都是线性表的例子，如人事档案表、职工工资表、学生成绩表、图书目录表、列车时刻表等。每一种这样的表通常都以记录登记的先后次序排列，或以关键字(即某个域的值)升序或降序排列。如职工工资表按职工号字段的升序排列，学生成绩表按学生号字段的升序排列，列车时刻表按开出时间字段的升序排列等。

线性表中的元素通常是变化的，但元素之间的逻辑关系应保持不变，当从一个位置上删除一个元素后，其后的所有元素都要依次前移一个位置，同样当向一个位置插入一个元素时，该位置的原有元素及后面的所有元素都要依次后移一个位置。这是线性表同集合在操作上的本质区别所在。从集合中删除一个元素后，不需要移动元素，只需要简单地用最后一个元素来填补空出的位置；向集合中插入元素不需要考虑插入位置，因而不需要移动任何元素，只要简单地放到表头或表尾即可。线性表与集合在元素取值上的区别是：集合中的元素值不允许重复，而线性表中的元素值允许出现重复，由此反映到插入算法上，不需要首先从线性表中查找是否存在被插入的元素。

同第 2 章讨论的集合的情况一样，线性表的长度是变化的，当向它插入一个元素后其长度就增加 1，当从中删除一个元素后其长度就减少 1。

线性表中的元素类型同集合一样，可以为任何一种数据类型。在定义集合 Set 接口类时采用的是根基类 java.lang.Object，因为任何其他的实际类型都自动继承根基类。而在本章中，也将采用系统提供的根基类 java.lang.Object，但为了能够进行线性表中元素(或关键字)大小的比较，实际的元素类型还应该实现系统提供的 java.lang.Comparable 接口并定义有 compareTo 成员方法，这样才能够利用 compareTo 方法来比较元素的大小，以便能够对线性表进行按值插入或排序等运算。在 Java 系统提供的类中，如简单类型的包装类和字符串类等，都实现了 Comparable 接口，即都带有 compareTo 方法的定义。同使用比较两个对象相等的 equals 方法一样，而用户能够调用 compareTo 方法，该方法的含义为：当调用对象大于参数对象的值时，则返回一个正整数；当调用对象小于参数对象的值时，则返回一个负整数；当调用对象等于参数对象的值时，则返回整数 0。若用户传递给线性表中元素对象的类型为用户自己定义的类，则定义该类时要实现 Comparable 接口，并在定义体中给出 compareTo 方法的定义。

3.1.2　线性表的抽象数据类型

线性表的抽象数据类型同样由数据和操作这两个部分组成。数据部分为一个采取任一种存储方法存储的线性表；操作部分与对集合的操作类似，包括插入、删除、查找、更新、

遍历输出等典型运算。利用这些典型运算，可以很容易编写出对线性表进行其他任何运算的算法。

线性表的抽象数据类型定义如下：

```
ADT List is                              //线性表的抽象数据类型的定义
    Data:
        采用任一种存储方法存储的一个线性表
    Operation:
        InitList();                      //创建一个线性表并初始化为空表
        value(pos);                      //返回线性表中第 pos 个元素的值
        add(obj,pos);                    //向第 pos 个位置插入一个新元素
        remove(pos);                     //删除第 pos 个元素并返回,若不存在返回空值
        find(obj,pos);                   //从第 pos 个元素起查找元素,返回元素序号
        modify(obj,pos);                 //用新值修改线性表中给定序号的元素值
        isEmpty();                       //判断线性表是否为空
        size();                          //返回线性表的长度
        nextOrder();                     //正序(即从前向后)遍历线性表中的每个元素
        preOrder();                      //反序(即从后向前)遍历线性表中的每个元素
        clear();                         //清除线性表中的所有元素,使之变为空表
        sort();                          //根据当前线性表排序生成新的有序表并返回
end List
```

同集合的抽象数据类型的定义一样，在线性表的抽象数据类型中也只给出了一些典型操作，用户在开发软件时可以根据实际需要增加或减少相关操作。线性表的抽象数据类型也需要采用 Java 语言中的接口类来描述，假定仍采用 List 作为接口名。在线性表接口类 List 中，只需要包含抽象数据类型中的操作声明部分（创建和初始化线性表的操作除外），不需要包含其数据部分，线性表的数据部分和对其进行初始化的操作，以及其他每个对线性表操作的具体定义应包含在实现线性表接口类 List 的相应的具体子类中。

线性表接口类 List 的定义如下：

```
public interface List
{
    Object value(int pos);                  //返回线性表中第 pos 个元素的值
    boolean add(Object obj,int pos);        //向第 pos 个位置插入一个元素
    Object remove(int pos);                 //删除第 pos 个元素并返回,若不存在返回空值
    int find(Object obj,int pos);           //从第 pos 个元素起查找元素,返回序号
    boolean modify(Object obj,int pos);     //修改线性表中给定序号的元素值
    boolean isEmpty();                      //判断线性表是否为空
    int size();                             //返回线性表的长度
    void nextOrder();                       //正序遍历线性表中的所有元素
    void preOrder();                        //反序遍历线性表中的所有元素
    void clear();                           //清除线性表中的所有元素,使之变为空表
    List sort();                            //根据当前线性表排序生成新的有序表并返回
}
```

3.1.3　线性表运算举例

例 3-1　假定一个整型数组 a={20,16,38,42,29},x=80,y=42,待操作的当前线性表 list 为空,则对当前线性表 list 进行的一组运算如下:

```
for(int i=0; i<a.length; i++) list.add(a[i],i+1);
     //向线性表 list 依次插入数组 a 中每个整型元素,list 为(20,16,38,42,29)
list.remove(2);          //删除 list 中第 2 个元素,结果变为(20,38,42,29)
list.add(x,3);           //向 list 的第 3 个位置插入 x 值,结果变为(20,38,80,42,29)
list.modify(33,4);       //把第 4 个元素修改为 33,结果变为(20,38,80,33,29)
list.nextOrder()         //从前向后依次输出 list 中的所有 5 个元素
list.preOrder();         //从后向前依次输出 list 中的所有 5 个元素
list.clear();            //清除 list 线性表中的所有元素,使之变为空表()
```

例 3-2　假定学生类(Student)的定义为:

```
public class Student implements Comparable   //实现接口 Comparable 用于比较
{
    String numb;                             //学号
    String name;                             //姓名
    int grade;                               //成绩

    public Student(String nu,String na,int gr)  {    //构造函数
        numb=nu; name=na; grade=gr;
    }
    public boolean equals(Object obj) {      //定义比较两学生相同的方法
        Student s=(Student)obj;
        return numb.equals(s.numb);          //假定通过对学号的比较来判断相等
    }
    public String toString() {               //定义用于输出的学生记录的字符串形式
        return numb+" "+name+" "+grade+"\n";
    }
    public int compareTo(Object obj) {       //定义比较两学生大小的方法
        Student s=(Student)obj;
        return grade-s.grade;                //假定通过对成绩的比较来判断大小
    }
}
```

假定以 Student 为元素类型的一个线性表 list 所含元素如表 3-1 所示。

表 3-1　学生成绩表

numb(学号)	name(姓名)	grade(成绩)	numb(学号)	name(姓名)	grade(成绩)
201	xuxk	73	204	tongy	90
202	weir	84	205	chaot	82
203	xucong	65			

下面首先给出具有 Student 类的 x、y、z 引用对象的定义并赋初值。

```
Student x=new Student("206","baojuan",80);     //给出一条新记录
Student y=new Student("201"," ",0);            //只给出学号用于查找
Student z=new Student("210","aabb",88);        //给出一条新记录
```

接着给出对线性表 list 进行的一组操作：

```
list.add(x,3);                                 //把 x 插入到线性表 list 的第 3 个元素位置
Student a1=(Student)list1.remove(2);           //删除第 2 个元素,返回该元素值
int b1=list.find(y,1);                         //从 list 的第 1 个元素起查找与 y 值相同
                                               //的元素,返回的元素序号为 1,被赋给 b1
list.modify(z,5);                              //用 z 的值重写第 5 个元素的值
list1.nextOrder();                             //正序输出 list 中的所有元素,如表 3-2 所示
List list2=list.sort();                        //对 list 进行排序所形成的线性表赋给 list2
list2.nextOrder();                             //正序输出 list2 中的所有元素,如表 3-3 所示
```

表 3-2　对线性表 list 操作后的结果

numb(学号)	name(姓名)	grade(成绩)
201	xuxk	73
206	baojuan	80
203	xucong	65
204	tongy	90
210	aabb	88

表 3-3　线性表 list2 中的所有元素

numb(学号)	name(姓名)	grade(成绩)
203	xucong	65
201	xuxk	73
206	baojuan	80
210	aabb	88
204	tongy	90

3.2　线性表的顺序存储和操作实现

同集合的顺序存储结构一样，线性表的顺序存储结构也是利用一个数组来实现的，利用具有 Object 元素类型的一维数组来存储线性表中的所有元素，另外，还要利用一个整型变量来存储线性表的当前长度，即一维数组所存储的线性表中当前包含的元素个数，以及利用一个整型常量来保存待定义数组的初始长度。这 3 个对象的定义假定为：

```
final int maxSize=10;                  //假定存储线性表的一维数组的初始长度为 10
private Object[] listArray;            //定义存储线性表的数组对象
private int length;                    //定义数组中所保存线性表的当前长度
```

线性表中的每个元素将依次被存入到一维数组 listArray 中，具体地说，线性表中第 1 个元素 a_1 被存入到 listArray[0]中，第 2 个元素 a_2 被存入到 listArray[1]中，以此类推，最后一个元素 a_n 被存入到 listArray[n－1]中。这样，线性表元素之间的线性有序关系，通过数组中下标位置的递增顺序自然地反映出来，即 listArray[i]元素的前驱元素是它的前一个下标位置上的元素 listArray[i－1]，后继元素是它的后一个下标位置上的元素 listArray[i＋1]；下标为 0 的元素为表头元素，它没有前驱，下标为 n－1 的元素为表尾元素，它没有后继。

一个线性表顺序存储结构示意图如图 3-1 所示，它与集合的顺序存储结构的示意图完全相同。

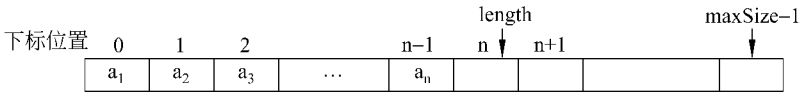

图 3-1　线性表的顺序存储结构的示意图

这里要注意的是：集合和线性表的顺序存储结构，都是从下标 0 位置开始存储元素的，因此，集合和线性表中的第 i 个元素，对应在数组中存储的下标位置为 i－1。若把下标为 0 的位置废弃不用，而从下标为 1 的位置开始存储元素，则元素在集合或线性表中的序号就与数组中存储的下标位置序号相同了。

为了把线性表当作一个整体对象，应定义为一个类，该类应包括线性表的采用顺序存储结构的数据定义部分和相应构造方法的定义部分，以及实现线性表接口类 List 中所有操作的成员方法的定义部分。假定采用顺序存储结构实现的线性表类所命名的标识符为 sequenceList，下面将依次给出并讨论顺序线性表类中的每个方法的定义。

1. 初始化线性表为空

在该类的构造函数中，初始化线性表为空，并为保存线性表的数组 listArray 分配存储空间。该构造函数有两个，一个为无参构造函数，另一个为带有数组初始长度参数的构造函数。这两个构造函数的具体定义如下。

```
public sequenceList()                    //无参的构造函数的定义
{
    length=0;                            //线性表初始为空,即长度为 0
    listArray=new Object[maxSize];       //数组初始长度为 maxSize 的值 10
}

public sequenceList(int n)               //带初始长度参数的构造函数的定义
{
    if(n<=0) {
        System.out.println("数组长度要大于 0,否则退出程序运行!");
        System.exit(1);
    }
    length=0;                            //线性表的初始长度为 0
    listArray=new Object[n];             //数组的初始长度为 n 的值
}
```

2. 得到线性表中第 pos 个元素的值

线性表中的第 pos 个元素被对应存储在 listArray 数组中下标为 pos－1 的位置上，此算法的 pos 的有效值应大于等于 1 和小于等于线性表的实际长度。当 pos 值有效时则返回 listArray[pos－1]元素的值，否则返回一个空值，表明 pos 值无效。

```
public Object value(int pos)                    //返回线性表中第 pos 个元素的值
{
    if(pos<1 || pos>length) {
        System.out.println("参数 pos 的值不合法,无法得到元素!");
        return null;                            //返回空值表示参数无效
    }
    return listArray[pos-1];                    //返回线性表中第 pos 个元素的值
}
```

3. 向线性表中给定位置上插入一个元素

向线性表中插入一个元素可以插入到表头，也可以插入到表尾，还可以插入到任何确定位置，这里假定插入到第 pos 个元素的位置，若参数 pos 值有效则插入后返回 true，否则无法进行插入则返回 false。此插入算法应按照下列步骤进行。

(1) 检查 pos 的值是否有效，pos 值应大于等于 1，同时要小于等于线性表的当前长度 length 加 1。若等于 length+1 则是把新元素插入到表尾元素的后一个位置上，成为新的表尾元素。若 pos 值无效，则返回 false，表明无法插入。

(2) 检查存储线性表的一维数组空间是否已被占满，若是则重新分配多一倍的存储空间，并完成原有线性表的复制。这样才能够进行下面的新元素插入操作。

(3) 为了空出第 pos 个元素的位置，以便放入新元素，需要把它及之后的所有元素均后移一个存储位置。注意，要从表尾元素开始依次向前，一个一个地后移元素值，若相反，则移动的元素将覆盖掉未移动的元素，结果会把第 pos 个元素的值移动到每个位置上，违反了算法的本意，导致出错。线性表的表尾元素所在的下标位置为 length－1，第 pos 个元素所在的下标位置为 pos－1。

(4) 把由参数 obj 所给的新元素赋给已空出的下标位置 pos－1 中，接着使线性表的长度增加 1。

(5) 返回 true，表示插入成功。

此算法具体描述为：

```
public boolean add(Object obj,int pos)
{                                               //向线性表的第 pos 个位置插入一个新元素
    if(pos<1 || pos>length+1) {                 //检查 pos 是否有效
        System.out.println("参数 pos 的值不合法,无法插入元素!");
        return false;                           //插入元素失败,返回 false
    }
    if(length==listArray.length) {              //对数组空间用完情况进行再分配
        Object[] p=new Object[length*2];        //新数组空间为原来的 2 倍
        for(int i=0; i<length; i++) p[i]=listArray[i];     //复制原线性表
        listArray=p;                            //使 listArray 指向新数组空间
    }
    for(int i=length; i>=pos; i--)              //为空出位置而移动元素
        listArray[i]=listArray[i-1];            //从表尾向前依次后移元素
    listArray[pos-1]=obj;                       //把 obj 赋给已空出的 pos-1 位置
```

```
    length++;                                    //线性表长度增加 1
    return true;                                 //返回 true,表示插入成功
}
```

在这个算法中,运行时间主要花费在第 3 步为空出插入位置所需的移动元素的次数。假定新元素插入位置 pos 是随机的,即取 1 ～ n＋1 之间的任一个值的概率都相同(n 表示线性表的长度),均为 1/(n＋1),则元素的平均移动次数等于在所有位置上插入一个元素的移动元素次数总和的平均值,即为$\frac{1}{n+1}\sum_{i=1}^{n+1}(n+1-i)=\frac{n}{2}$。所以此算法的平均时间复杂度为 O(n)。当然若在第 n＋1 的位置上插入,这属于最好的情况,此时不需要移动任何元素,其时间复杂度为 O(1)。

4. 从线性表中删除给定位置上的元素

从线性表中删除元素,可以删除任何位置上的元素,也可以按照给定的值或关键字删除对应的元素,这里假定要删除第 pos 个位置上的元素,pos 的有效取值范围在 1 和 length 之间,若超出此范围则无法得到被删除的元素,应返回空值表示删除失败。当 pos 值有效时,应返回被删除的元素,并需要将后面的所有元素前移一个位置,以保证元素存储的连续性。此删除算法应按照以下步骤进行。

(1) 检查所给的位置参数 pos 是否有效,若无效则给出提示信息并返回空值。

(2) 把线性表中第 pos 个位置上的元素暂时保存起来,以便返回。该位置元素在 listArray 数组中的下标存储位置为 pos－1。

(3) 把删除位置后面的所有元素均前移一个下标位置,首先前移的是第 pos＋1 个位置上的元素,占用已空出的第 pos 个元素的存储位置,第 pos＋1 个元素的位置空出后,再前移第 pos＋2 个位置上的元素,以此类推。

(4) 使线性表的当前长度值减 1。

(5) 返回已暂存起来的被删除元素。

此算法具体描述为:

```
public Object remove(int pos)
{                              //删除线性表中第 pos 个位置上的元素并返回,若不存在返回空值
    if(pos<1 || pos>length) {
        System.out.println("参数 pos 的值不合法,无法删除元素!");
        return null;                             //删除元素失败返回空
    }
    Object x=listArray[pos-1];                   //暂存第 pos 个位置的元素
    for(int i=pos; i<=length-1; i++)             //第 pos 个位置以后的元素依次前移
        listArray[i-1]=listArray[i];
    length--;                                    //线性表长度减 1
    return x;                                    //删除成功返回被删除元素
}
```

在这个算法中,运行时间主要花费在第 3 步为填补被删除元素的空出位置所需的移动元素的次数上。假定参数 pos 的值是任意的,在有效取值范围 1 ～ n 之间是等概率分布

的(n表示线性表的长度)，即为1/n，则元素的平均移动次数等于在所有位置上删除一个元素的移动元素次数总和的平均值，即为 $\frac{1}{n}\sum_{i=1}^{n}(n-i)=\frac{n-1}{2}$。所以此算法的平均时间复杂度为O(n)。当然若是要删除第n个位置上的表尾元素，这属于最好的情况，此时不需要移动任何元素，其时间复杂度为O(1)。

5. 从线性表中指定位置开始按值查找对应的第一个元素并返回所在位置

从线性表中查找元素分为顺序查找和二分查找两种。二分查找只适应于按值或关键字排列的、顺序存储的有序线性表，而顺序查找适应于任何线性表。顺序查找的过程很简单，它从给定的位置开始(一般从表头或表尾)，用待查值依次向后或向前同每个元素值进行比较，直到查找成功返回位置序号，若比较完所有元素没有找到待查元素，则返回－1表示查找失败。

在顺序存储的线性表上，从第pos个位置开始向后顺序查找与待查值obj相同的第一个元素的算法描述为：

```
public int find(Object obj,int pos)
{                                    //从第 pos 个位置的元素起顺序查找元素,返回位置序号
    if(pos<1 || pos>length) {                    //检查 pos 值是否有效
        System.out.println("参数 pos 的值不合法,无法查找元素!");
        System.exit(1);                          //退出程序运行,表示 pos 参数错误
    }
    for(int i=pos-1; i<length; i++)              //第 pos 个元素位置对应下标为 pos-1
        if(listArray[i].equals(obj)) return i+1;          //查找成功返回序号
    return -1;                                            //返回-1 表示查找失败
}
```

此算法的运行时间，主要用在for循环上。假定pos取值为1，即从表头元素开始进行顺序查找，并且线性表中的每个元素都有同等的概率等于待查值obj，则查找成功时元素比较次数的平均值为 $\frac{1}{n}\sum_{i=1}^{n}i=\frac{n+1}{2}$，算法的平均时间复杂度为O(n)。若pos的取值不是1，则要小于这个平均值。

6. 修改线性表中给定位置的元素值

此操作要求把线性表中给定位置上的元素值用新的元素值替换，若替换成功则返回true，否则返回false。此算法带有两个参数，一个为Object类型的对象obj，另一个为整型对象pos，用pos给出待修改的元素位置，用obj给出新的修改值。若pos的取值在有效范围1～n之内，则修改后返回true，若超出此范围则返回false，表示修改失败。此算法的具体描述为：

```
public boolean modify(Object obj,int pos)
{                                        //修改线性表中给定位置上的元素值
    if(pos<1 || pos>length) {            //检查 pos 的取值是否有效
        System.out.println("参数 pos 的值不合法,无法修改元素!");
```

```
            return false;                    //返回 false,表示因 pos 参数错误而修改失败
        }
        listArray[pos-1]=obj;                //用 obj 的值更新第 pos 个元素的值
        return true;                         //返回 true,表示修改元素成功
    }
```

此算法的时间复杂度为 O(1)。

7. 判断线性表是否为空

此算法要求若线性表为空则返回 true,否则返回 false。当线性表的长度为 0 时则为空,否则不为空。

```
public boolean isEmpty()                     //判断线性表是否为空
{
    return length==0;
}
```

8. 求出线性表长度

此算法返回当前线性表的长度,即 length 域的值。

```
public int size()                            //返回线性表的长度
{
    return length;
}
```

9. 正序遍历线性表中的所有元素

此算法要求按照线性表中每个元素的逻辑顺序从表头向表尾遍历所有元素,就是依次访问一遍每个元素。假定访问一个元素的操作,使用输出元素的值代之,则此算法的具体描述为:

```
public void nextOrder()
{                                            //正序遍历输出线性表中的每个元素
    for(int i=0; i<length; i++)              //访问每个元素时假定以输出元素值代之
        System.out.println(listArray[i].toString());
}
```

10. 反序遍历线性表中的所有元素

此算法要求按照线性表中逻辑顺序的相反方向从表尾向表头遍历所有元素。假定访问一个元素的操作,仍使用输出元素的值代之,则此算法的具体描述为:

```
public void preOrder()
{                                                //反序遍历线性表中的每个元素
    for(int i=length-1; i>=0; i--)               //访问每个元素时以输出元素值代之
        System.out.println(listArray[i].toString());
}
```