数据结构

Ch2 线性表

第一章回顾

- 数据结构
  - 数据的逻辑结构
    - 线性结构
      - 线性表
      - 栈（特殊线性表）
      - 队列（特殊线性表）
      - 字符串、数组、广义表
    - 非线性结构
      - 树形结构
      - 图形结构
  - 数据的存储结构
    - 顺序存储
    - 链式存储
  - 数据的运算：　检索、排序、插入、删除、修改等

Ch2.线性表

1. 概述
2. 线性表的顺序存储结构
3. 线性表的链式存储结构
4. 线性表的应用实例

Ch2.线性表

- 简介：
  - 线性表的定义及特点
  - 线性表的顺序存储结构
  - 线性表的链式存储结构
  - 线性表的应用实例:用线性表表示集合、一元多项式
- 重点：
  - 线性表的基本概念和术语
  - 线性表的顺序表示和链式表示方法及其上的基本操作
  - 相关算法的时间复杂度分析
- 难点：线性表的链式表示和基本操作的实现

Ch2.线性表：概述——线性表定义

- 线性表(Linear List):性质相同的数据元素构成的有限序列。
  - 如(3,5,56,67,88),(a,b,c,d,e,f)都是线性表
  - 更复杂的线性表中，一个数据元素可以由若干个数据项组成（此时可称为记录），含有大量记录的线性表称为文件。如:学生基本信息表:

学号	姓名	性别	出生日期	民族	专业	…
20102430101	赵光	男	1992.6.3	汉	计算机应用	…
20102430102	钱伟	男	1992.2.20	回	计算机应用	…
20102430311	孙丽	女	1992.12.8	汉	软件工程	…

- 线性表的数据元素（记录）可由若干数据项组成。
- 线性表中的数据元素类型可以是多种多样的，但同一表中的元素必定具有相同特性，即属于同一数据对象

Ch2.线性表：概述——线性表术语

线性表通常可以表示为（$a_1, a_2, \ldots, a_n$）

- **线性表的长度（表长）**：线性表中元素的个数（n）
- **空表**：n=0时，称为空表
- **位序**：在非空的线性表中，每个元素都有一个确定的位置，如（$a_1, a_2, \ldots, a_i, \ldots, a_n$）中，i 为数据元素$a_i$在线性表中的位序，$a_1$为起始节点，$a_n$为终端节点
- 称a_i是a_{i+1}的**直接前驱**元素，a_{i+1}是a_i的**直接后继**元素

Ch2.线性表：概述——线性结构的特点

1

- 存在惟一的一个被称作“第一个”的数据元素（a_1）；
- 存在惟一的一个被称作“最后一个”的数据元素（a_n）

2

相邻数据元素之间存在序偶关系，若将线性表记为（$a_1,a_2,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n$），则$a_{i-1}$领先于$a_i$

a_{i-1}是a_i的直接前驱元素

a_{i+1}是a_i的直接后继元素

3

除最后一个之外，线性表中每个数据元素均只有一个直接后继

4

除第一个之外，线性表中的每个数据元素均只有一个直接前驱

线性表案例

一元多项式的运算：实现两个多项式的加、减、乘运算

$P_n(x) = p_0 + p_1x + p_2x^2 + \ldots + p_nx^n$

线性表P = $(p_0, p_1, p_2, \ldots, p_n)$

例如：$P_n(x) = 10 + 5x - 4x^2 + 3x^3 + 2x^4$

指数(下标i)	0	1	2	3	4
系数p[i]	10	5	-4	3	2

线性表案例

$$R_n(x) = P_n(x) + Q_m(x)$$

线性表R = $(p_0 + q_0, p_1 + q_1, p_2 + q_2, \ldots, p_m + q_m, p_{m+1}, \ldots, p_n)$

稀疏多项式：$S(x) = 1 + 3x^{10000} + 2x^{20000}$

下标	0	1	2
指数	0	10000	20000
系数p[i]	1	3	2

浪费存储空间
如何处理？

稀疏多项式的运算

非零多项式的数组表示

(a) $A(x) = 7 + 3x + 9x^8 + 5x^{17}$　(b) $B(x) = 8x + 22x^7 - 9x^8$

下标i	0	1	2	3
系数a[i]	7	3	9	5
指数	0	1	8	17

下标i	0	1	2
系数b[i]	8	22	-9
指数	1	7	8

$P_n(x) = p_1x^{e1} + p_2x^{e2} + \ldots + p_mx^{em}$

线性表P = ((p1, e1), (p2, e2), ..., (pm, em))

线性表A=((7, 0), (3, 1), (9, 8), (5, 17)),
线性表B=((8, 1), (22, 7), (-9, 8))

稀疏多项式的运算

线性表A=((7, 0), (3, 1), (9, 8), (5, 17)),
线性表B=((8, 1), (22, 7), (-9, 8))

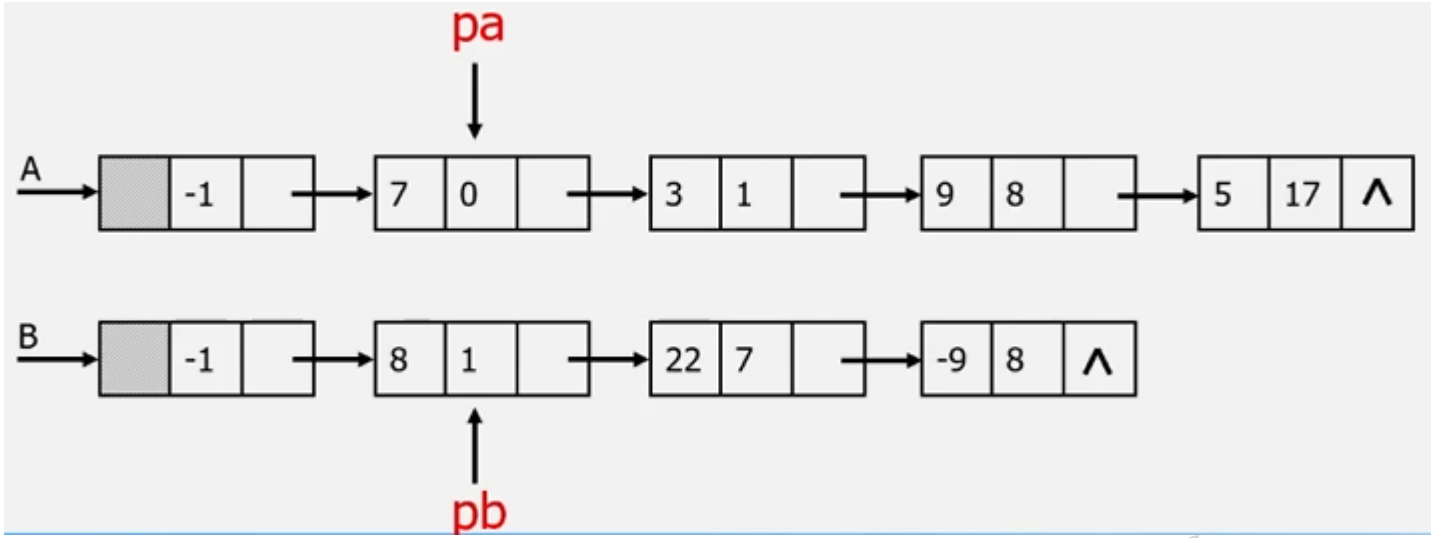
- 创建一个新数组c
- 从头遍历比较a和b的每一项
  - 指数相同，对应系数相加，若其和不为0，则在c中增加一个新项
  - 指数不相同，则将指数较小的项复制到c中

一个多项式已遍历完毕时，将另一个剩余项一次复制到c中

稀疏多项式的运算

- 顺序存储结构的问题：
  - 存储空间分配不灵活 → 链式存储结构
  - 运算的空间复杂度高

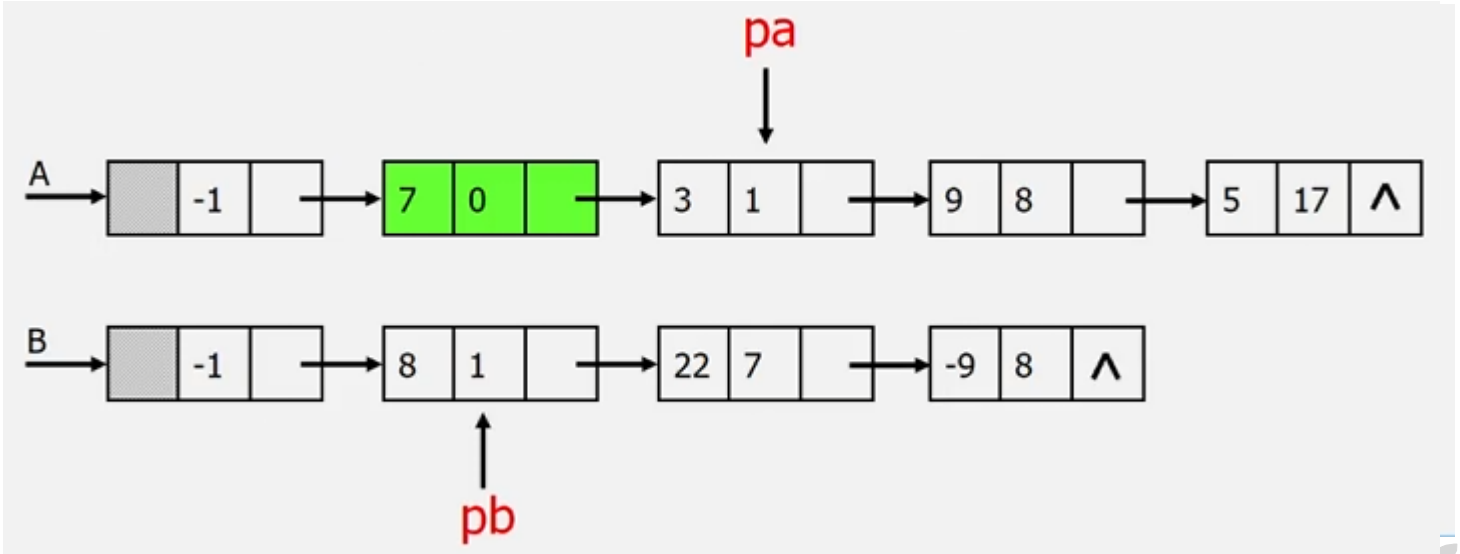
$A(x) = 7 + 3x + 9x^8 + 5x^{17}$　　$B(x) = 8x + 22x^7 - 9x^8$



稀疏多项式的运算

- 顺序存储结构的问题：
  - 存储空间分配不灵活 → 链式存储结构
  - 运算的空间复杂度高

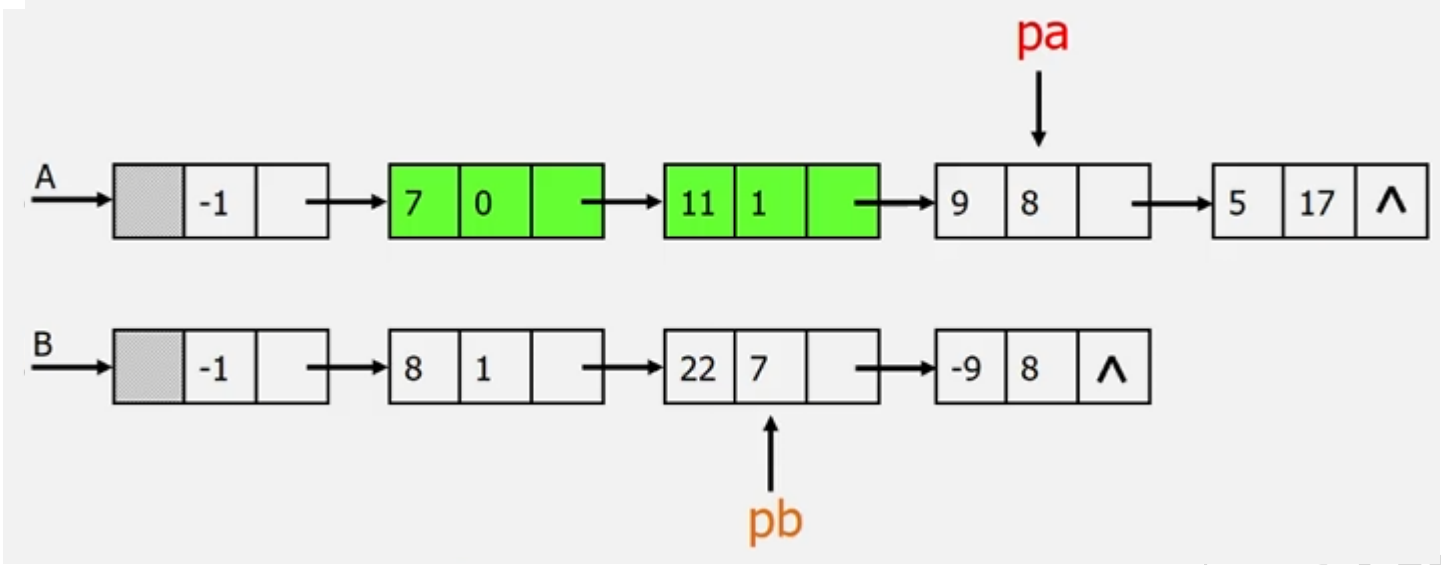
$A(x) = 7 + 3x + 9x^8 + 5x^{17}$　　$B(x) = 8x + 22x^7 - 9x^8$



稀疏多项式的运算

- 顺序存储结构的问题：
  - 存储空间分配不灵活 → 链式存储结构
  - 运算的空间复杂度高

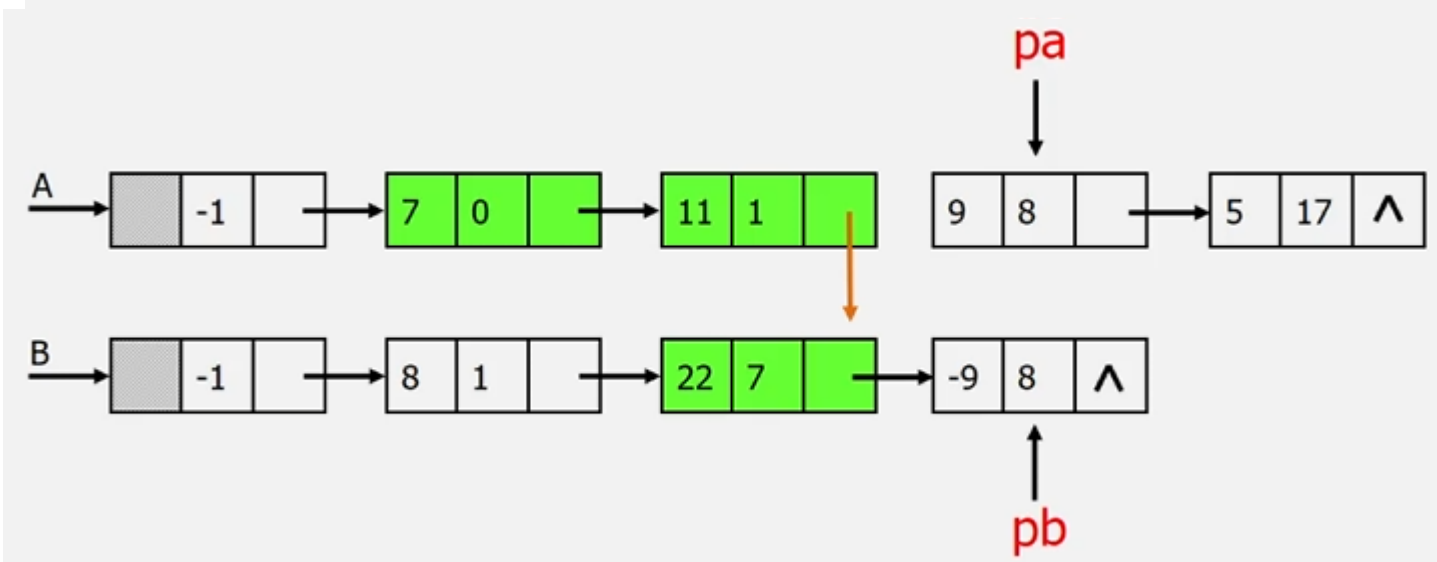
$A(x) = 7 + 3x + 9x^8 + 5x^{17}$　　$B(x) = 8x + 22x^7 - 9x^8$



稀疏多项式的运算

- 顺序存储结构的问题：
  - 存储空间分配不灵活 → 链式存储结构
  - 运算的空间复杂度高

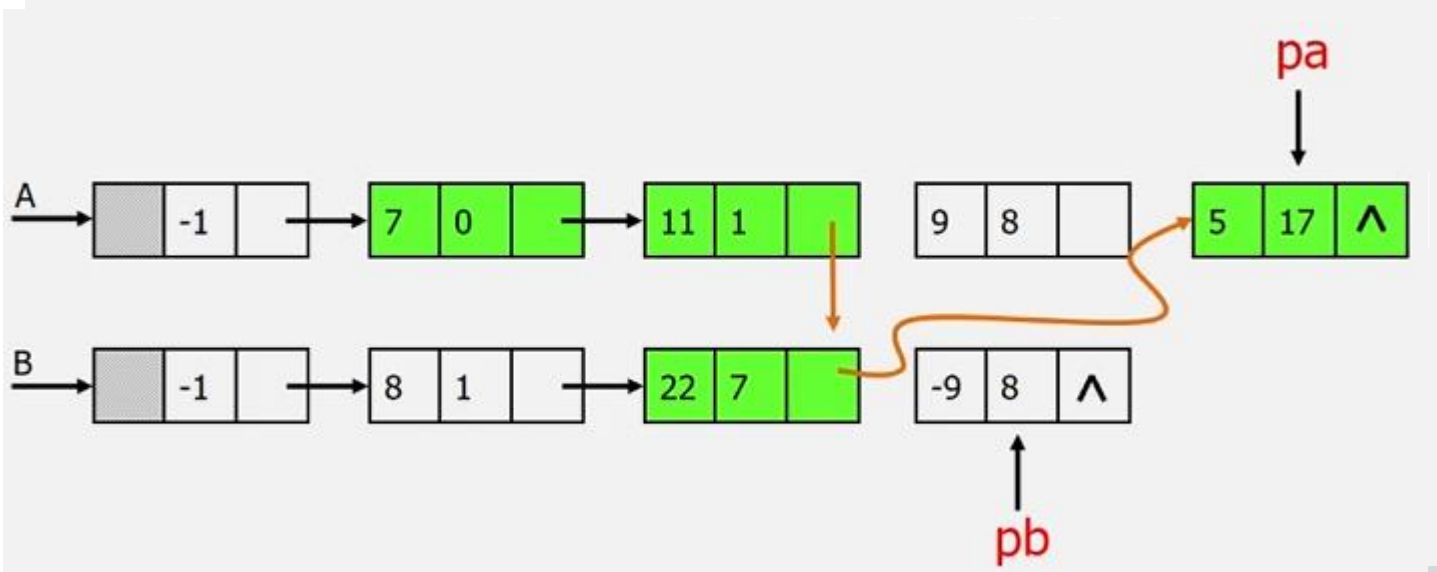
$A(x) = 7 + 3x + 9x^8 + 5x^{17}$　　$B(x) = 8x + 22x^7 - 9x^8$



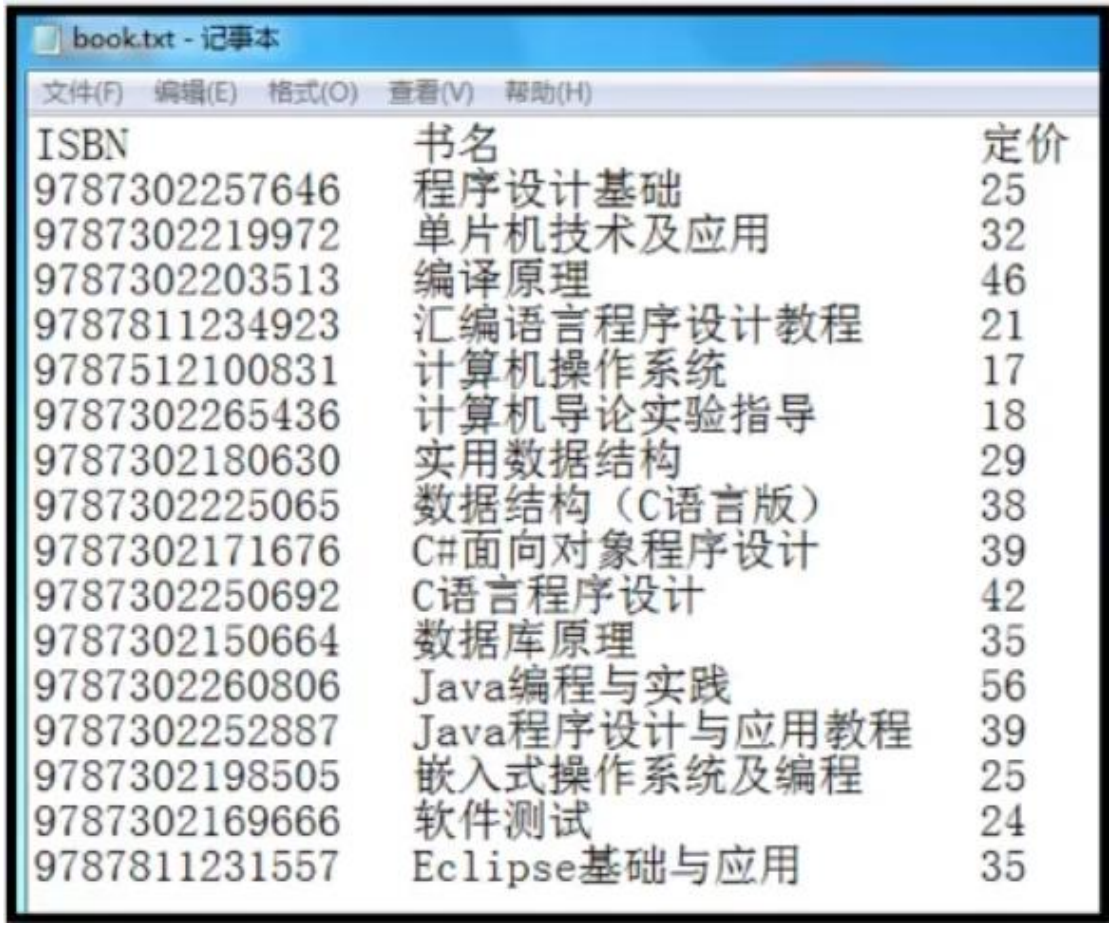
稀疏多项式的运算

- 顺序存储结构的问题：
  - 存储空间分配不灵活 → 链式存储结构
  - 运算的空间复杂度高

$A(x) = 7 + 3x + 9x^8 + 5x^{17}$　　$B(x) = 8x + 22x^7 - 9x^8$



线性表案例

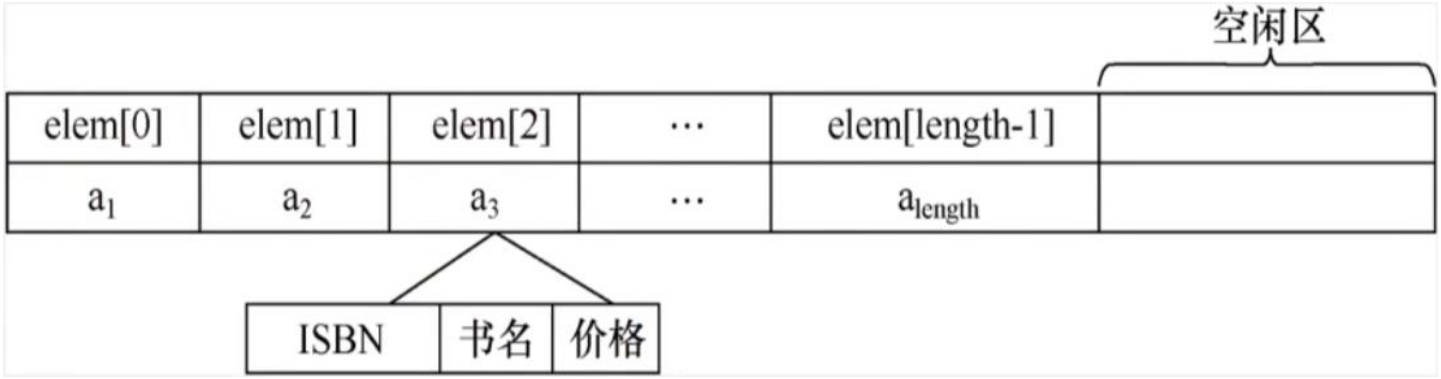


需要的功能：
(1) 查找　(2) 插入
(3) 删除　(4) 修改
(5) 排序　(6) 计数

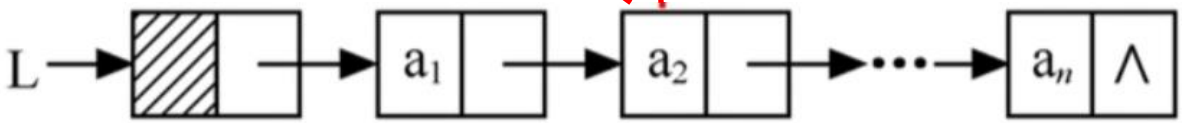
- 图书表抽象为线性表
- 表中图书抽象为线性表中的数据元素

线性表案例

- 图书顺序表



- 图书链表



- ✓ 选择适当的存储结构
- ✓ 实现此存储结构上的基本操作
- ✓ 利用基本操作完成功能

总结

- 线性表中数据元素的类型可以为简单类型或复杂类型
- 许多实际应用问题所设计的基本操作有很大相似性，不应为每个具体应用单独编写一个程序
- 从具体应用中抽象出共性的逻辑结构和基本操作（抽象数据类型），然后实现其存储结构和基本操作

补充：操作算法中用到的预定义常量和类型

```
//函数结果状态代码
#define   TRUE       1
#define   FALSE      0
#define   OK         1
#define   ERROR      0
#define   INFEASIBLE -1
#define   OVERFLOW   -2
//Status 是函数的类型，其值是函数结果状态代码
typedef   int  Status;
typedef   char ElemType;
```

Ch2.线性表：概述——线性表应用

- 需求：线性表 LA 和 LB 分别表示集合 A 和 B，求 A=A∪B。
- 解决方案：把存在于LB且不存在于LA的元素插入LA。

示例

- 算法描述：

```
void set_union(List &La，List Lb) {
    int La_len，Lb_len，i;
    ElemType e ;
    La_len=ListLength(La);
    Lb_len=ListLength(Lb);
    for(i=1; i<=lb_len; i++){
        ①从Lb中取第 i 个元素e
        ②如果在La中不存在与e相等的元素
        ③则在La的表尾插入e
  }//for
}// set_union
```

算法中可不写变量声明

GetElem(Lb,i,e);

if(!LocateElem(La,e,equal)

ListInsert(La,++La_len,e);

Ch2.线性表：概述——线性表应用2

•需求：把2个非递减有序的线性表LA和LB归并为非递减有序的线性表LC。

示例

```
void MergeList(List La, List Lb, List &Lc){//...
        InitList(Lc);
        i=j=1; k=0;  //初始化
        La_len=ListLength(La); Lb_len=ListLength(Lb); //分别求La、Lb表长
        while(i<=La_len && j<=Lb_len){
           GetElem(La,i,ai); GetElem(Lb,j,bj);
           //把 ai、bj 中较小的那个插入到 Lc 的表尾
           if(ai<=bj) { ListInsert(Lc,++k,ai); ++i; }
           else { ListInsert(Lc,++k,bj); ++j; }
        }
        while(i<=La_len) { GetElem(La,i++,ai); ListInsert(Lc,++k,ai); } //...
        while(j<=Lb_len) { GetElem(Lb,j++,bj); ListInsert(Lc,++k,bj); } //...
}
```

书写规范

Ch2.线性表

1 概述

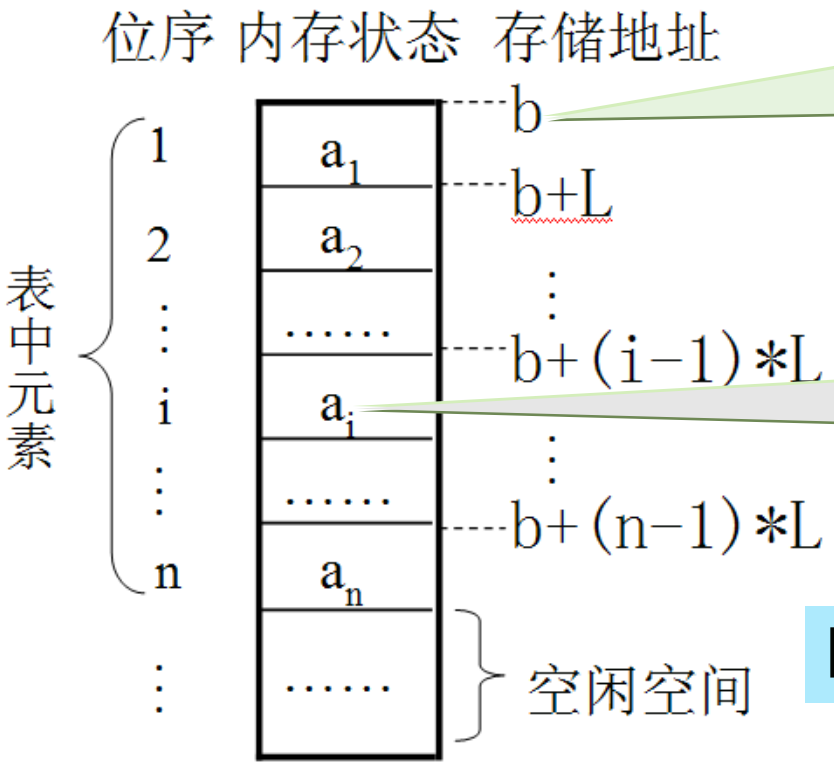
2 线性表的顺序存储结构

3 线性表的链式存储结构

4 线性表的应用实例

Ch2.线性表：线性表的顺序存储结构——定义

- 线性表的顺序表示：用一组地址连续的存储单元依次存储线性表的数据元素。用顺序存储结构表示的线性表又称为顺序表。



第1个元素的存储地址LOC(a1)为线性表的起始地址（基地址）

设线性表的每个元素占L 个存储单元

$$LOC(a_i)=LOC(a_1)+(i-1)*L$$

Ch2.线性表：线性表的顺序存储结构——特点

- 顺序表特点
  - 元素的存储位置能反映元素间的逻辑关系，逻辑上相邻的数据元素在物理位置上也相邻。
  - 只要确定顺序表在内存中的起始位置，表中的任何元素都可随机存取，顺序表是一种随机存取的存储结构

数组

Ch2.线性表：线性表的顺序存储结构——类型定义

- 顺序表的实现：由于线性表的长度可变，而数组长度不可动态定义，所以用动态分配的一维数组来实现

常量的值可根据实际需要自行定义

- 顺序表的动态分配存储结构

```
#define LIST_INIT_SIZE 100  //顺序表存储空间的初始分配量
#define LISTINCREMENT 10    //顺序表存储空间的分配增量
typedef struct {
    ElemType *elem;   //存储空间的基地址
    int length;       //顺序表的当前长度
    int listsize;     //数组存储空间的长度
}SqList;
```

Ch2.线性表：线性表的顺序存储结构——类型定义

$P_n(x) = p_1x^{e1} + p_2x^{e2} + \ldots + p_mx^{em}$

线性表P = ((p1, e1), (p2, e2), ..., (pm, em))

```
#define MAXSIZE 1000     //多项式可能达到的最大长度

typedef struct   {       //多项式非零项的定义
    float p;             //系数
    int   e;             //指数
}Polynomial;

typedef struct{
 Polynomial *elem;       //存储空间的基地址
  int length;            //多项式中当前项的个数
}SqList;                 //多项式的顺序存储结构类型为SqList
```

Ch2.线性表：线性表的顺序存储结构——类型定义

book.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

ISBN	书名	定价
9787302257646	程序设计基础	25
9787302219972	单片机技术及应用	32
9787302203513	编译原理	46
9787811234923	汇编语言程序设计教程	21
9787512100831	计算机操作系统	17
9787302265436	计算机导论实验指导	18
9787302180630	实用数据结构	29
9787302225065	数据结构（C语言版）	38
9787302171676	C#面向对象程序设计	39
9787302250692	C语言程序设计	42
9787302150664	数据库原理	35
9787302260806	Java编程与实践	56
9787302252887	Java程序设计与应用教程	39
9787302198505	嵌入式操作系统及编程	25
9787302169666	软件测试	24
9787811231557	Eclipse基础与应用	35

```
#define MAXSIZE 10000
                        //图书表可能达到的最大长度

typedef struct  { //图书信息定义
    char no[20];                //图书ISBN
    char name[50];              //图书名字
    float price;                //图书价格
}Book;

typedef struct{
   Book *elem;  //存储空间的基地址
   int length;     //图书表中当前图书个数
}SqList;     //图书表的顺序存储结构类型为SqList
```

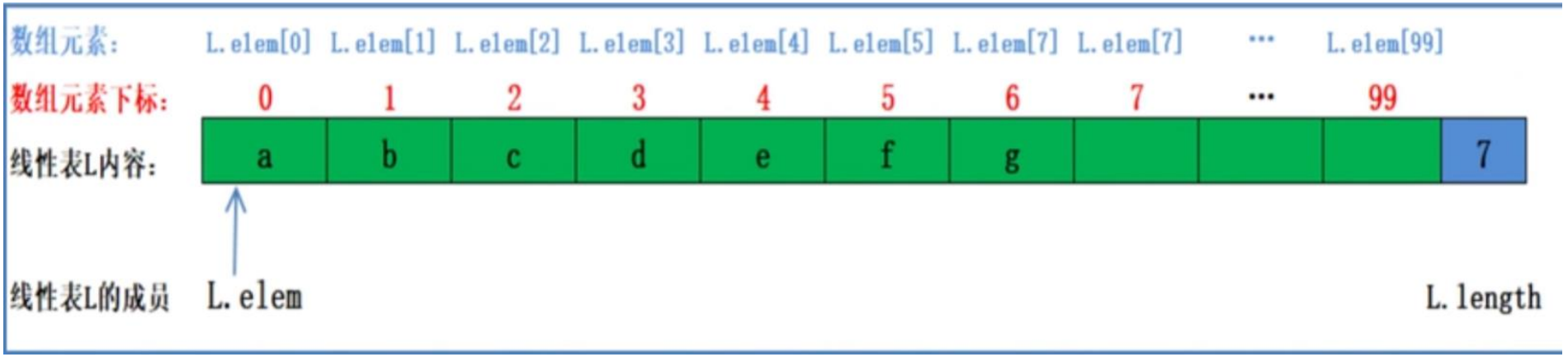
顺序表示意图

```
#define MAXSIZE 100
typedef struct{
  ElemType *elem;
  int length;
} SqList;   //定义顺序表类型
```

就像：

```
int a;   //定义变量a，a是int型
```

```
SqList L;   //定义变量L，L是SqList这种类型的，L是个顺序表
```



Ch2.线性表：概述——线性表定义

- 线性表(Linear List):性质相同的数据元素构成的有限序列。
  - 如(3,5,56,67,88),(a,b,c,d,e,f)都是线性表
  - 更复杂的线性表中，一个数据元素可以由若干个数据项组成（此时可称为记录），含有大量记录的线性表称为文件。如:学生基本信息表:

学号	姓名	性别	出生日期	民族	专业	…
20102430101	赵光	男	1992.6.3	汉	计算机应用	…
20102430102	钱伟	男	1992.2.20	回	计算机应用	…
20102430311	孙丽	女	1992.12.8	汉	软件工程	…

- 线性表的数据元素（记录）可由若干数据项组成。
- 线性表中的数据元素类型可以是多种多样的，但同一表中的元素必定具有相同特性，即属于同一数据对象

Ch2.线性表：概述——ADT线性表

ADT List {

数据对象：D={a_i | a_i∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1={< a_{i-1} , a_i > | a_{i-1} , a_i ∈D, i=2,3,...,n }

基本操作：

InitList(&L); //构造一个空的线性表L

ListEmpty(L); //判断线性表L是否是空表，若是，则返回TRUE，否则返回FALSE

ListLength(L); //返回线性表L的长度

GetElem(L, i, &e) ; //用e返回线性表L的第i个数据元素的值

LocateElem(L, e, compare());//在线性表L中查找第一个和元素e满足compare关系
//的元素，若找到则返回其位序；否则返回0

PriorElem(L, e, &pre_e); //用pre_e返回线性表L中元素e的直接前驱

NextElem(L, e, &next_e); //用next_e返回线性表L中元素e的直接后继

ListInsert(&L, i, e) ; //将数据元素e插入到线性表L的第i个数据元素之前

ListDelete(&L, i, &e); //删除线性表L的第i个数据元素，并将其值用e返回

ListTraverse(L, visit()); //依次对线性表L中的每个元素调用visit进行访问

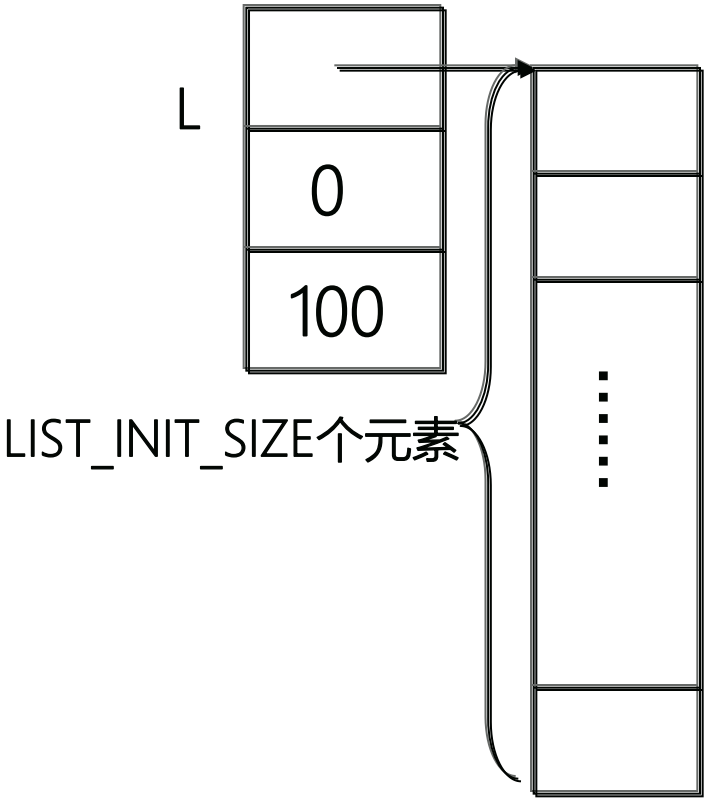
ClearList(&L); //重置线性表L为空表

DestroyList(&L); //销毁线性表L，可能的话释放其空间

}ADT List

Ch2.线性表：线性表的顺序存储结构——初始化

```
Status InitList(SqList &L){
    L.elem=(ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
    if(!L.elem) exit(OVERFLOW);
    L.length=0;
    L.listsize=LIST_INIT_SIZE;
    return OK;
}
```



函数调用：

```
SqList L;
if(InitList(L)==OK)
  printf("%d",L.length);
……
```

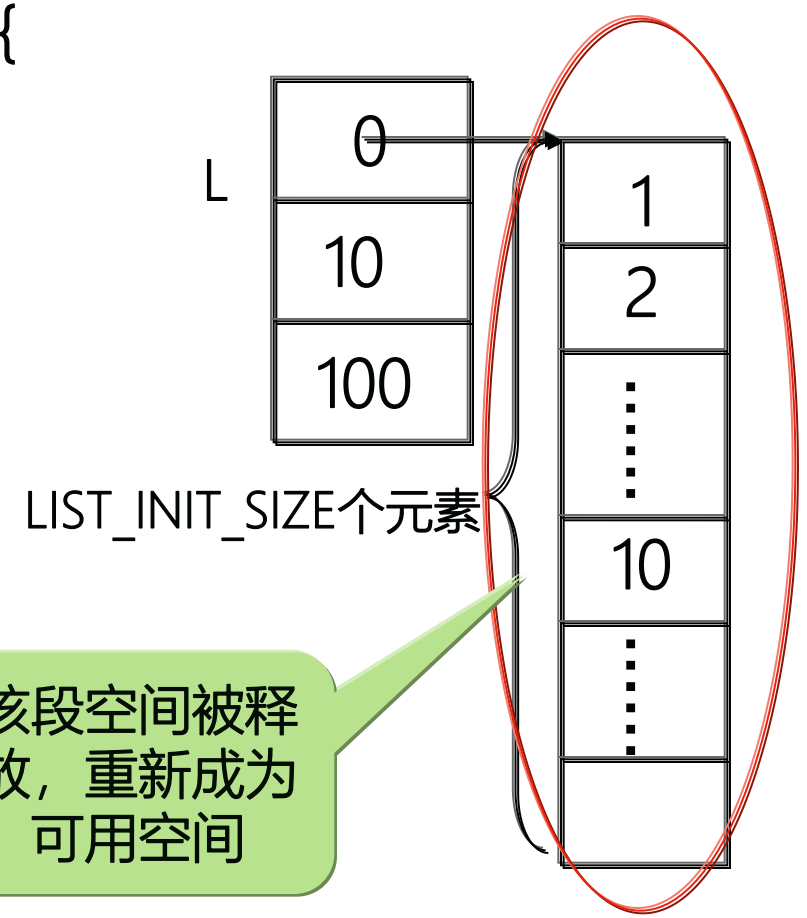
输出：0

Ch2.线性表：线性表的顺序存储结构——销毁

```
void DestroyList(SqList &L) {
  if(L.elem) free(L.elem);
  L.elem=NULL;
}
```

函数调用：

```
SqList L;
InitList(L);
    ……
DestroyList(L);
```



Ch2.线性表：线性表的顺序存储结构——清空

```
void ClearList(SqList &L)
{
        L.length=0;
}
```

Ch2.线性表：线性表的顺序存储结构——判空

顺序表的判空操作：

```
Status ListEmpty(SqList L)
{
    if(L.length==0) return TRUE;
    else return FALSE;
}
```

函数调用：if(!ListEmpty(L)) {//……}

Ch2.线性表：线性表的顺序存储结构——求表长

顺序表的求表长操作：

```
int ListLength(SqList L)
{
    return L.length;
}
```

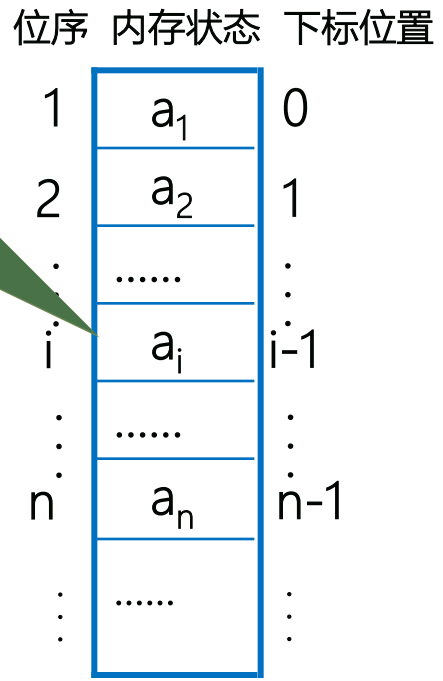
函数调用：

printf(“The length is: %d\n”,ListLength(L));

Ch2.线性表：线性表的顺序存储结构——取值

```
Status GetElem(SqList L, int i, ElemType &e){
    if(i<1 || i>L.length) return ERROR;
    e=L.elem[i-1];
    return OK;
}
```

特别注意，表中的第i个元素是数组的第i-1个分量！



函数调用：

```
if(GetElem(L, i, e))
  printf("The i'th element is:%d",e);
else
  printf("位置不合法。");
```

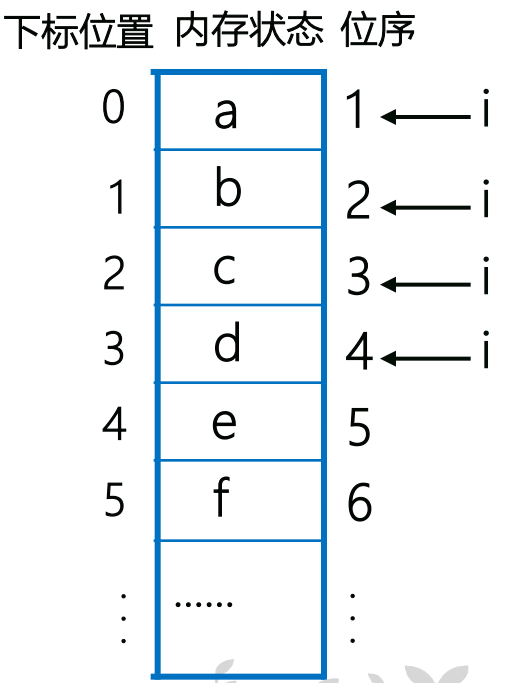
前6个基本操作的时间复杂度均为O(1)

Ch2.线性表：求前驱、后继的操作可在定位基础上完成

```
int LocateElem (SqList L, ElemType e)
{
    i=1;
    while( i<=L.length && L.elem[i-1]!=e)  i++;
    if(i<=L.length) return i;
    else return 0;
}
```

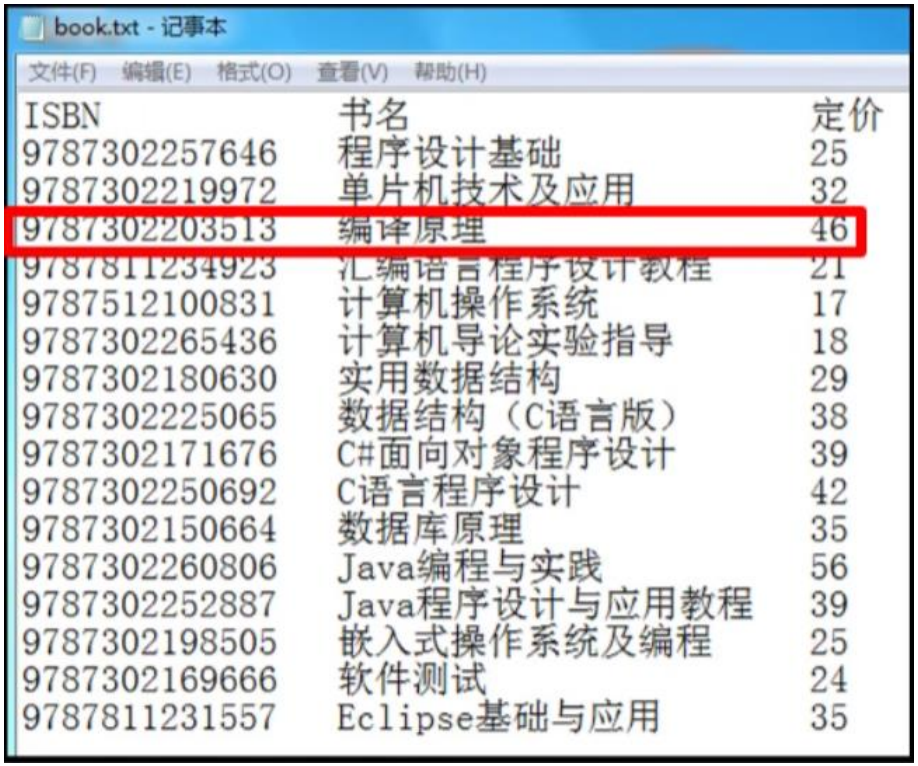
调用定位函数：
k=LocateElem(L,'d');
则k=4

该算法的时间复杂度均为O(L.length)



Ch2.线性表：线性表的顺序存储结构——定位

- 按值查找
- 例如：在图书表中，按照给定的书号进行查找，确定是否存在该图书
- 如果存在：输出是第几个元素
- 如果不存在：返回0



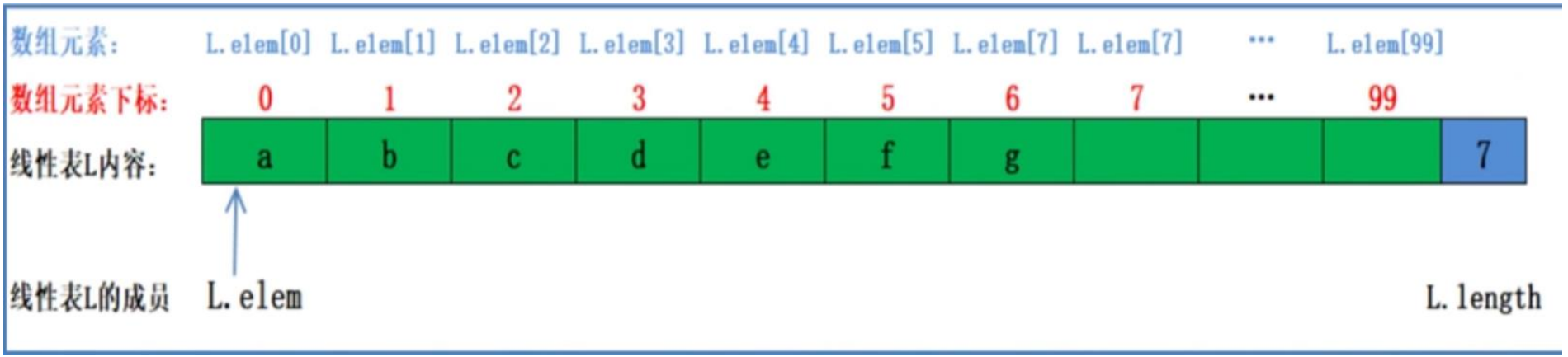
Ch2.线性表：线性表的顺序存储结构——定位

- 顺序表的查找：
  - 在线性表中查找与指定值e相同的数据元素的位置
  - 从表的一段开始，逐个进行记录的关键字和给定值的比较。找到则返回该元素的位置符号；未找到则返回**0**

```
int LocateELem(SqList L, ElemType e){
//在线性表L中查找值为e的数据元素，返回其序号（是第几个元素）
    for (i=0;i< L.length;i++)
        if (L.elem[i]==e) return i+1; //查找成功，返回序号
    return 0; //查找失败，返回0
}
```

Ch2.线性表：线性表的顺序存储结构——定位

- 顺序表的查找算法分析：
  - 查找的关键操作为：记录的关键字和给定值的比较。
  - 基本操作：**if(L.elem[i]==e)**



Ch2.线性表：线性表的顺序存储结构——定位

- 顺序表的查找算法分析：
  - 平均查找长度ASL (Average Search Length):
    - 为确定记录在表中的位置，需要与给定值进行比较的关键字的个数的期望值
  - 对含有n个记录的表，查找成功时：

$$ASL = \sum_{i=1}^{n} P_i C_i$$

  - C_i：找到第i个记录需比较的次数
  - P_i：第i个记录被查找的概率

Ch2.线性表：线性表的顺序存储结构——定位

- 顺序查找的平均长度：

$$ASL = P_1 + P_2 + \cdots + (n-1)P_{n-1} + nP_n$$

  - 假设每个记录的被定位概率相等：$P_i = 1/n$
  - 则：

$$ASL = \sum_{i=1}^{n} P_i C_i = \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}$$

Ch2.线性表：线性表的顺序存储结构——定位

```
int LocateElem (SqList L, ElemType e, Status
(*compare)(ElemType, ElemType))
{
    i=1;  p=L.elem;
    while(i<=L.length && !(*compare)(*p++,e)) ++i;
    if(i<=L.length) return i;
    else return 0;
}
```

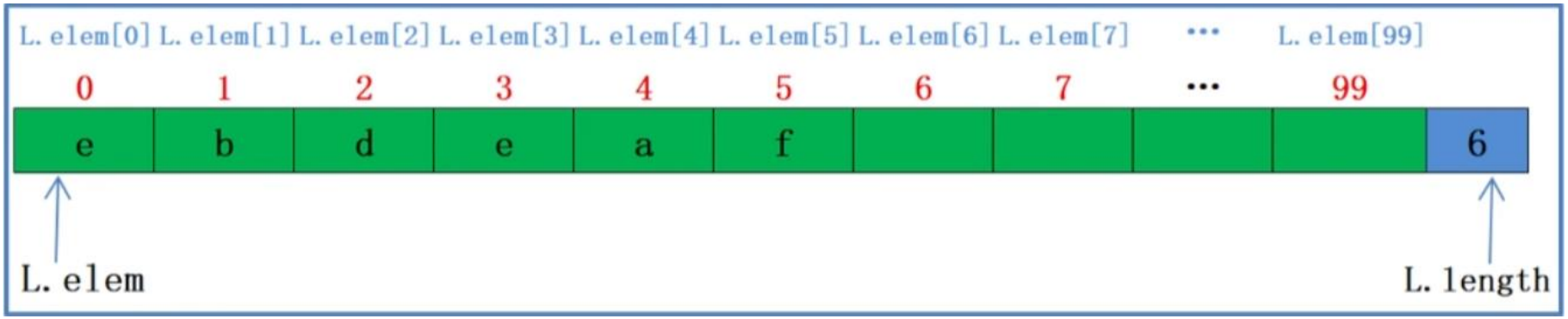
定义比较函数：

```
Status GT(ElemType a,ElemType b){
    if(a>b) return TRUE;
    else return FALSE;
}
```

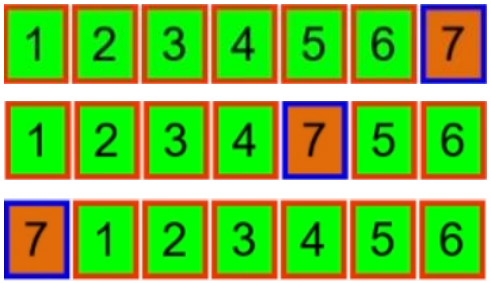
调用定位函数：**k=LocateElem(L,a,GT);**

Ch2.线性表：线性表的顺序存储结构——插入

- 顺序表的插入

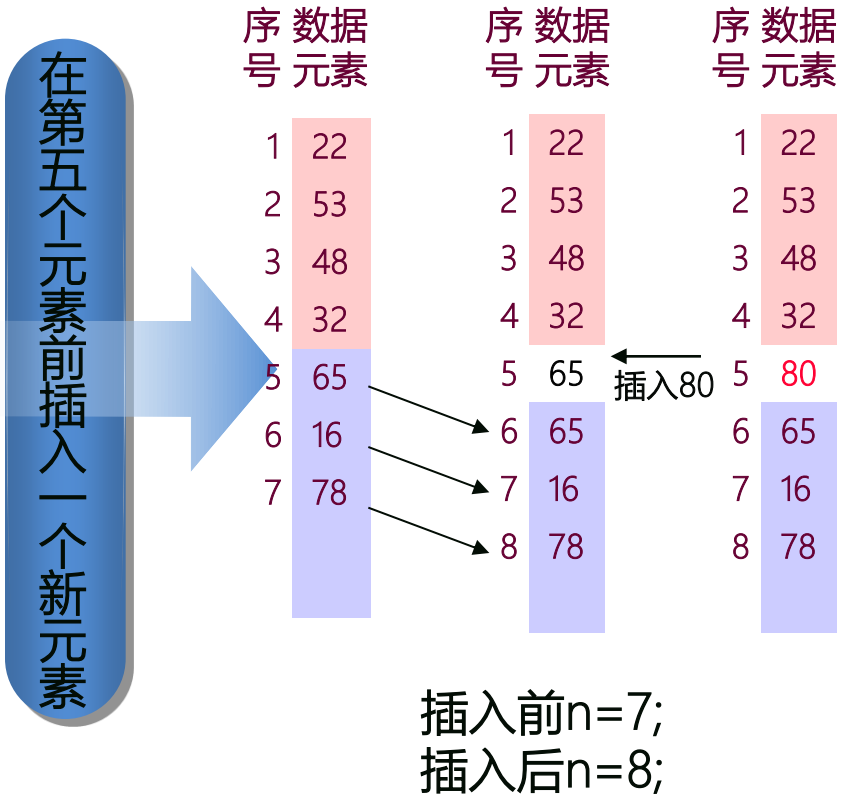


- 插入不同位置的算法演示：
  - 插入位置在最后
  - 插入位置在中间
  - 插入位置在最前面



Ch2.线性表：线性表的顺序存储结构——插入

顺序表的插入操作：



插入前n=7;
插入后n=8;

- 线性表的插入操作：
  - 原线性表:$(a_1,\ldots a_{i-1},a_i,\ldots,a_n)$
  - 插入后：$(a_1,\ldots a_{i-1},e,a_i,\ldots,a_n)$
- 插入时需做的操作有：
  - 将第n个到第i个共n-i+1个数据元素依次向后移动一个位置。
  - 在表的第i个位置上放入e
  - 表长加1

Ch2.线性表：线性表的顺序存储结构——插入

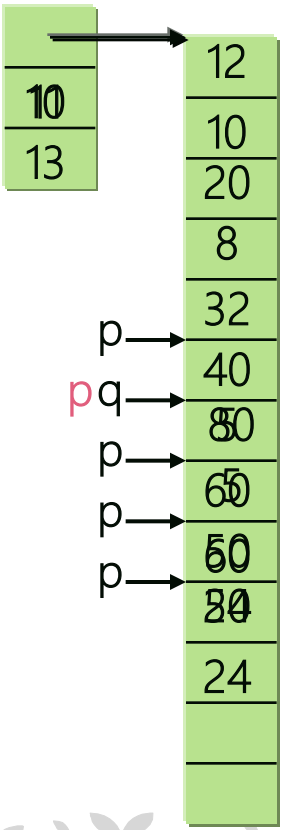
- 算法思想：
  - 判断插入位置i是否合法
  - 判断顺序表的存储空间是否已满，若已满则返回ERROR
  - 将第n至第i位的元素依次向后移动一个位置，空出第i个位置
  - 将要插入的新元素e放入第i个位置
  - 表长+1，插入成功返回OK

```
Status ListInsert
    //插入元素e到
    if(i<1 ||i>L.length
    if(L.length=
     { … }//重新分配空间
    q=&L.elem[i-1];
    for(p=&(L.elem[L.length-1];p>=q;--p)
         *(p+1)=*p;
    *q=e;
    ++L.length;
    return OK;
}
```

```
L.elem=(ElemType *) realloc(L.elem,
(L.listsize+LISTINCREMENT)*sizeof(ElemType) );
if(!L.elem) exit(OVERFLOW);
L.Listsize += LISTINCREMENT;
```

ERROR;
L.listize)

在第7个元素前插入80



Ch2.线性表：顺序表的插入——算法效率分析

- 在顺序表中第i个位置前插入元素的时间主要耗费在移动 n-i+1 个元素上
- 设p_i是在第i个元素之前插入一个元素的概率，不失一般性，假定在线性表的任何位置上插入元素都是等概率的，则

$$p_i = \frac{1}{n+1} \qquad (1 \le i \le n+1)$$

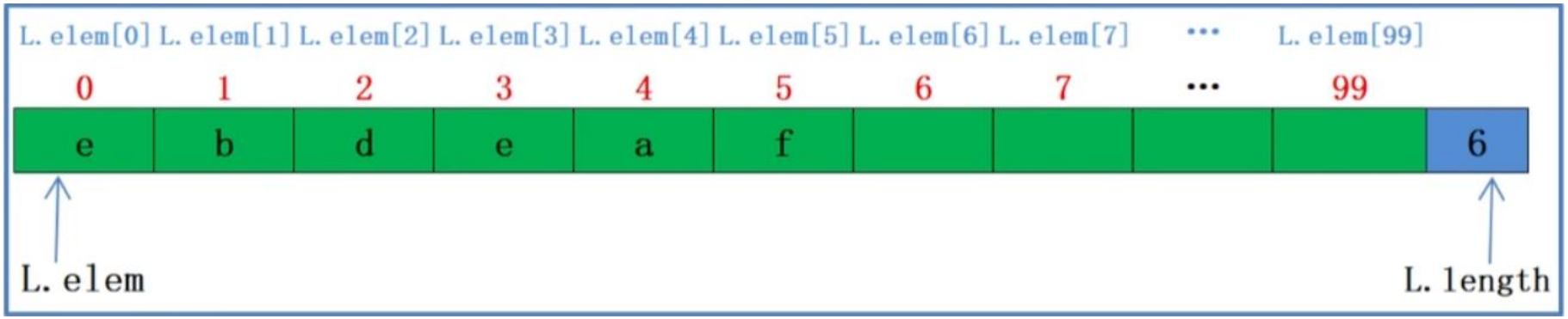
- 在长度是n的顺序表中插入一个元素时所需移动元素次数的数学期望值(平均次数)为:

$$E_{ins} = \sum_{i=1}^{n+1} p_i (n-i+1) = \frac{1}{n+1} \sum_{i=1}^{n+1} (n-i+1) = \frac{n}{2}$$

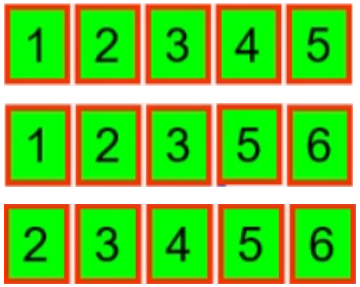
- 顺序表的插入算法的时间复杂度为 **O(n)**

Ch2.线性表：线性表的顺序存储结构——删除

- 顺序表的删除

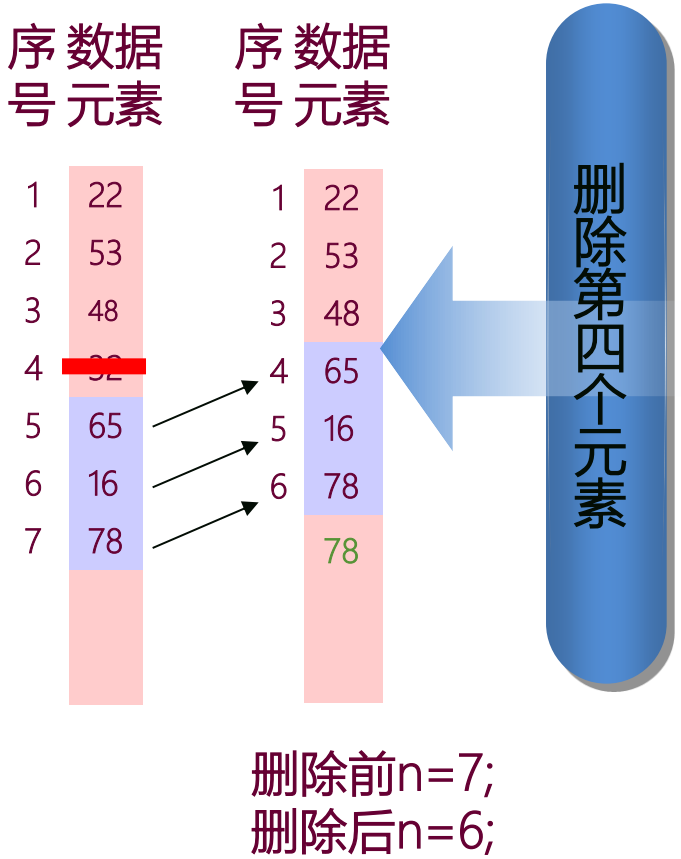


- 删除不同位置的算法演示：
  - 删除位置在最后
  - 删除位置在中间
  - 删除位置在最前面



Ch2.线性表：线性表的顺序存储结构——删除

- 删除第i个元素的操作:
  - 删除前:$(a_1,\ldots a_{i-1},a_i,a_{i+1},\ldots,a_n)$
  - 删除后:$(a_1,\ldots a_{i-1},a_{i+1},\ldots,a_n)$
- 删除时需做的操作有：
  - 将第i+1到第n个共n-i个数据元素依次向前移动一个位置。
  - 表长减1



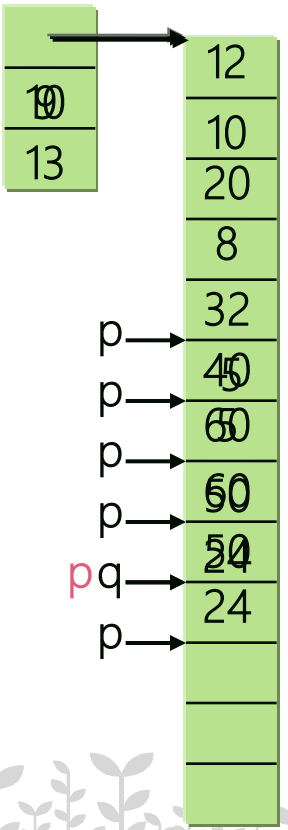
删除前n=7;
删除后n=6;

Ch2.线性表：线性表的顺序存储结构——删除

- 算法思想：
  - 判断删除位置i是否合法（合法值为1≤i ≤ n）
  - 将要删除的元素保留在e中
  - 将第i+1至第n位的元素依次前移动一个位置
  - 表长-1，删除成功返回OK

Ch2.线性表：线性表的顺序存储结构——删除算法

```
Status ListDelete(SqList &L,int i,ElemType &e){
    //删除顺序表L中的第i个元素，其值由e返回;
    if(i<1 ||i>L.length) return ERROR;
    p=&L.elem[i-1];
    e=*p;
    q=L.elem+L.length-1;
    for(++p;p<=q;++p)
        *(p-1)=*p;
    --L.length;
    return OK;
}
```



Ch2.线性表：顺序表的删除——算法效率分析

- 在顺序表中删除第i个元素的时间主要耗费在移动 n-i 个元素上
- 假设q_i是删除第i个元素的概率，不失一般性，假定在线性表的任何位置上删除元素都是等概率的，则

$$q_i = \frac{1}{n} \qquad (1 \le i \le n)$$

- 在长度为n的顺序表中删除一个元素时所需移动元素次数的数学期望值

$$E_{del} = \sum_{i=1}^{n} q_i (n-i) = \frac{1}{n} \sum_{i=1}^{n} (n-i) = \frac{n-1}{2}$$

- 顺序表的删除算法的时间复杂度为 O(n)。

顺序表的遍历操作：

```
Status ListTraverse(SqList L,Status (*visit)(ElemType))
{
  for(p=L.elem; p<L.elem+L.length; p++)
      if( !visit(*p) ) return ERROR;
  return OK;
}
```

定义访问函数：

```
Status print(ElemType e){
      printf(e);
      return OK;
}
```

可如下调用定位函数：

```
if(ListTraverse(L,print)) printf("OK");
else printf("ERROR");
```

补充：实现一个具体应用的方法

- 利用我们自己定义的数据类型实现一个具体应用的2种方法
  - 使用已实现的数据类型（如线性表）定义变量，调用基本操作完成要做的运算
  - 直接在自定义的数据类型（如顺序表）上实现问题所要求的功能

Ch2.线性表：线性表的顺序存储结构——顺序表应用


- 需求： 把2个非递减有序的顺序表LA和LB归并为非递减有序的线性表LC。

```
void MergeList_Sq(SqList la,SqList lb,SqList &lc){//算法2.7
    pa=la.elem;  pb=lb.elem;
    lc.listsize=lc.length=la.length+lb.length;
    pc=lc.elem = new ElemType[lc.listsize];
    pa_last=la.elem+la.length-1;   pb_last=lb.elem+lb.length-1;
    while(pa<=pa_last && pb<=pb_last) {
        if(*pa<=*pb) *pc++=*pa++;
        else *pc++=*pb++;
    }
    while(pa<=pa_last) *pc++=*pa++;
    while(pb<=pb_last) *pc++=*pb++;
}
```

时间复杂度分析

Ch2.线性表：线性表的顺序存储结构——小结

- 特点：逻辑关系上相邻的两个元素在物理位置上也相邻
- 优点：可以随机存取表中任一元素；大部分操作都比较容易实现；存储利用率高（关系隐含在存储结构里，不需显式表示）
- 缺点：不知道表的规模的情况下，2个常量的大小不太好确定；顺序表在进行插入和删除操作时，需要移动大量元素，浪费大量时间。

- 适用场合：表中元素变动不大，但需要快速存取元素；或存储空间需求预先知道。