第二章 线性表

实际上，我们经常会遇到有序对象序列的组织与管理问题，所以本章讨论线性表的抽象定义，研究基于顺序存储和链式存储的线性表实现方法；并探讨线性表的基本操作：插入元素和删除元素。

第二章 线性表

2.1 线性表的类型定义

2.2 线性表的顺序表示和实现

2.3 线性表的链式表示和实现

2.4 一元多项式的表示和实现

2.1 线性表的类型定义

一. 线性表概念

- 线性表是n个数据元素的有限序列
- 线性数据结构的特点
  - 数据同一性，同一个线性表的数据属同一类数据对象
  - 顺序性，数据之间存在序偶关系
  - 在任意位置进行插入和删除操作

$$(a_1,\ a_2,\ \ldots a_{i-1},\ a_i,\ a_{i+1},\ \ldots a_{n-1},\ a_n)$$

其中，a_i是表中元素，i表示元素a_i的位置，n是表的长度

2.2 线性表的类型定义

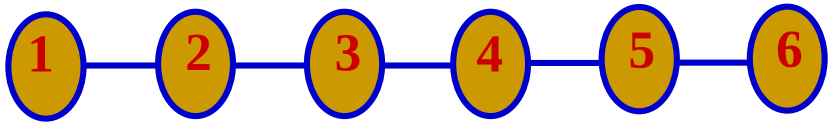
一. 线性表概念

- 数据同一性
  - 线性表中的元素具有相同的特性，属于同一数据对象，如：
    - 26个字母的字母表：（A, B, C, D, …, Z）
    - 近期每天的平均温度：(30℃， 28℃， 29℃, …)

2.1 线性表的类型定义

一. 线性表概念

- 数据的顺序性
  - 存在惟一的一个被称作“第一个”的数据元素(如“1”)
  - 存在惟一的一个被称作“最后一个”的数据元素(如“6”)
  - 除第一个元素外，每个数据元素均只有一个直接前驱
  - 除最后一个元素外，每个数据元素均只有一个直接后继(next)(如“1”的next是“2”，“2”的next是“3”)



2.1 线性表的类型定义

二. 线性表的ADT定义

ADT List{

数据对象：数据元素同属一个集合

数据关系：序偶关系

基本操作：

Init创建、Destroy销毁、

Clear清空、Empty是否为空、

Length取表长度、Get取表元素、Locate查找元素

Prior取元素前驱、Next取元素后继

Insert插入元素、Delete删除元素

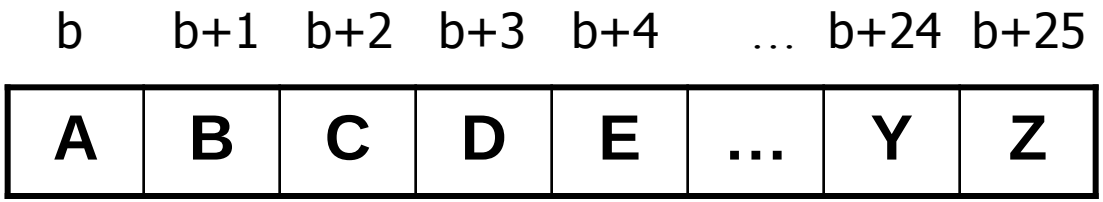
Traverse遍历表

}

2.2 顺序表

一. 顺序表概念

- 顺序表是线性表的顺序存储表示，用一组地址连续的存储单元依次存储线性表的数据元素。



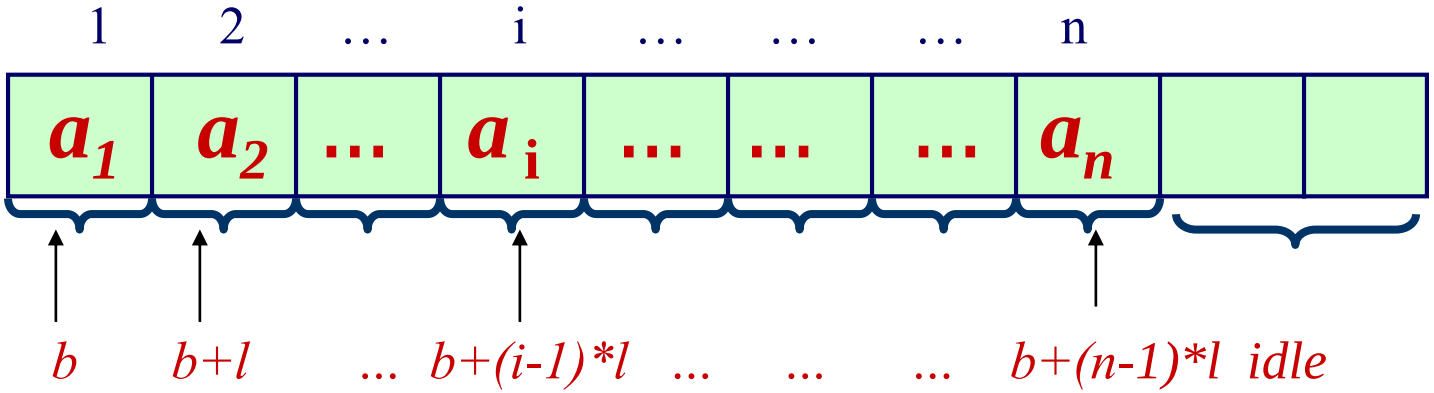
2.2 顺序表

二. 顺序表的数据位置

- 顺序表数据元素的位置：

  $LOC(a_i) = LOC(a_{i-1}) + l$

  $LOC(a_i) = LOC(a_1)+(i-1)*l$　　**l 表示元素占用的内存单元数**



2.2 顺序表

三. 顺序表的定义

- 采用C语言中动态分配的一维数组表示顺序表

```
#define LIST_INIT_SIZE   100  // 线性表存储空间的初始分配量
#define LISTINCREMENT    10   // 线性表存储空间的分配增量

Typedef struct {
     ElemType  *elem;          // 存储空间基址
     int       length;         // 当前长度
     int       listsize;       // 当前分配的存储容量(元素数)
} Sqlist;
```

2.2 顺序表

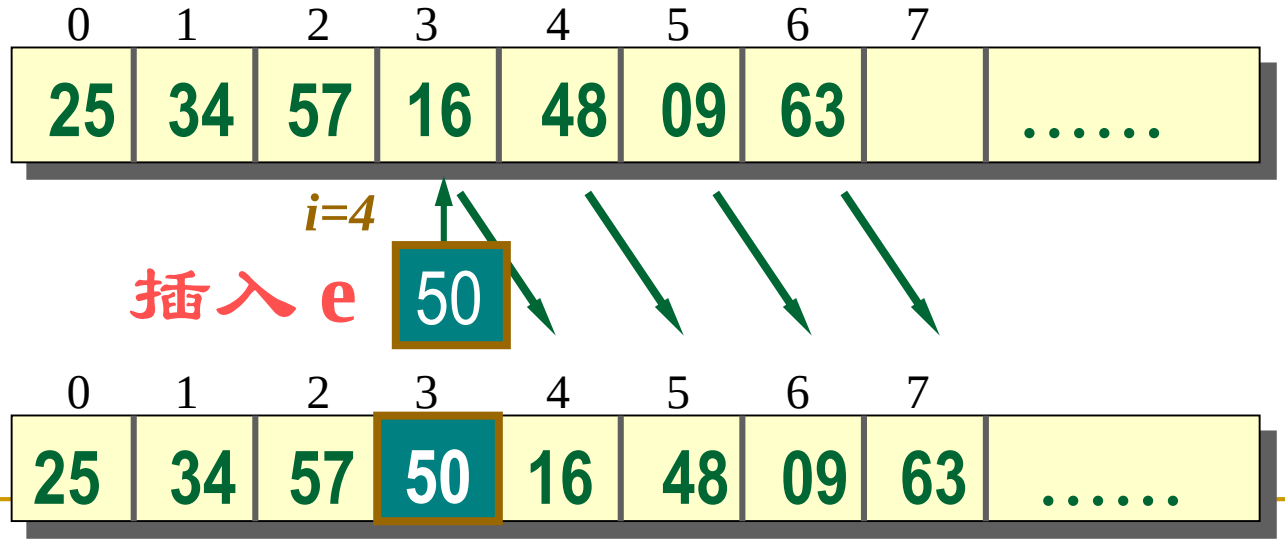
三. 顺序表的创建

```
Status InitList_Sq(Sqlist &L) {
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
     if (!L.elem) exit(OVERFLOW);                    // 存储分配失败
     L.length = 0;                                   // 空表长度为0
     L.listsize = LIST_INIT_SIZE;                    // 初始存储容量
     return OK;
} // InitList_Sq
```

2.2 顺序表

四. 顺序表的插入

- 给出插入的顺序表对象、位置、数据
  - 在顺序表的第i-1个数据元素和第i个数据元素之间插入一个新的数据元素
  - 操作包括后移、插入、长度+1
  - 例如，在第3个元素与第4个元素之间插入新元素e，需要将最后元素n至第4元素(共7-4+1)都向后移一个位置，长度加1



2.2 顺序表

四. 顺序表的插入

```
Status ListInsert_Sq(Sqlist &L, int i, ElemType &e) {
        if (i<1 || i>L.length+1) return ERROR;
        if (L.length >= L.listsize) {
           newbase = realloc(L,L.listsize+LISTINCREMENT)*sizeof(ElemType);
           if (!newbase) exit(OVERFLOW);
           L.elem = newbase;
           L.listsize += LISTINCRMENT; }          // 以上皆为准备阶段
        q = &(L.elem[i-1]);                       // 找到插入位置
        for (p=&(L.elem[L.length-1]); p>=q; --p) *(p+1) = *p;   // 右移
        *q = e;
       ++L.length;
       return OK;
} // ListInsert_Sq
```

2.2 顺序表

四. 顺序表插入的时间复杂度

- 在顺序表中第i个元素前插入一个元素，需要向后移动元素个数为：n-i+1

平均移动元素次数的期望值为：

$$E_{is} = \sum_{i=1}^{n+1} p_i * (n-i+1)$$

其中，当插入位置等概率时，$p_i=1/(n+1)$，因此：

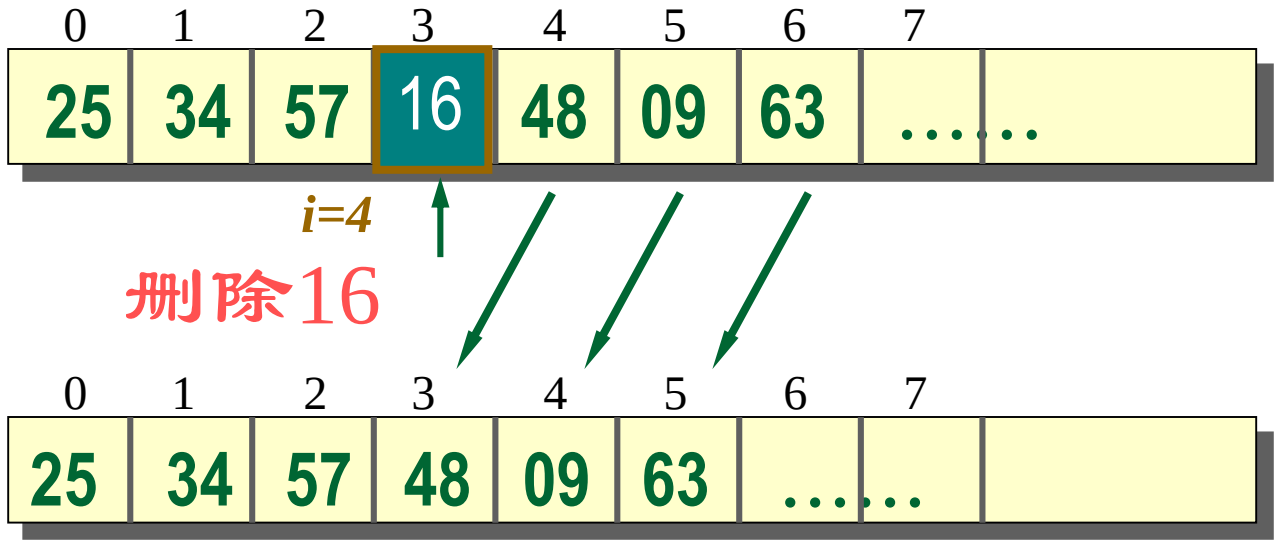
$$E_{is} = \sum_{i=1}^{n+1} [1/(n+1)] * (n-i+1) = n/2$$

- 顺序表插入操作的时间复杂度为O(n)

2.2 顺序表

五. 顺序表的删除

- 给出删除的顺序表对象、位置
  - 指将顺序表的第i个数据元素删除
  - 操作包括删除、前移、长度-1
  - 例如将第4个元素删除，需要将最后元素n至第5元素(共7-4)都向前移一个位置，长度减1



2.2 顺序表

五. 顺序表的删除

```
Status ListDelete_Sq(Sqlist &L, int i, ElemType &e) {
    if ( i<1  ||  i>L.length)   return ERROR;
    p = &(L.elem[i-1]);                    // 找到要删除的元素位置
    e = *p;
    q = L.elem + L.length –1;              // 找到最后一个元素位置
    for (++p; p<=q; ++p)
        *(p-1) = *p;        // 左移
    --L.length;             // 表长减1
    return OK;
} // ListDelete_Sq
```

2.2 顺序表

五. 顺序表删除的时间复杂度

- 在顺序表中删除第i个位置上的元素，需要向前移动元素个数为：n-i，

  平均移动元素次数的期望值为：

$$E_{dl} = \sum_{i=1}^{n} q_i \times (n-i)$$

  其中，当删除位置等概率时，$q_i=1/n$，因此：

$$E_{dl} = \sum_{i=1}^{n} [1/n] \times (n-i) = (n-1)/2$$

- 顺序表删除操作的时间复杂度为O(n)

2.2 顺序表

五. 顺序表的合并

- 合并两个递增有序的队列，生成一个新的有序队列
  - 已知两个顺序表a和b，合并成顺序表c，三者都是递增有序

算法开始

1. 设定指针pa、pb、pc分别指向顺序表a、b、c的起始位置
2. 循环，条件是pa和pb都没到末尾

   比较pa和pb指向表a和表b的元素

   。。。。。。。
3. 把表a或表b的剩余元素复制到表c

算法结束

2.2 顺序表

六、顺序表的其它操作

- 查找第i个位置的元素值
- 检索元素所在位置
- 得到表长
- 置空表
- 销毁表

2.2 顺序表

七. 顺序表的优缺点

- 优点：
  - 元素可以随机存取

    元素位置可用一个简单、直观的公式表示并求取
- 缺点：
  - 在作插入或删除操作时，需要移动大量元素

    因此，引入链表，减少移动操作。

练习

一. 已知一顺序表包含数值11,22,33,44,55,66,77,88,99,111共10个元素，若执行以下操作，计算共移动了多少次，并给出执行后的结果

a. 在第5位置插入数值101

b. 在第8位置插入数值202

c. 删除第3位置元素