数据结构与算法

Data Structure and Algorithms

西安交通大学自动化系

蔡忠闽 周亚东

第二章　线 性 表

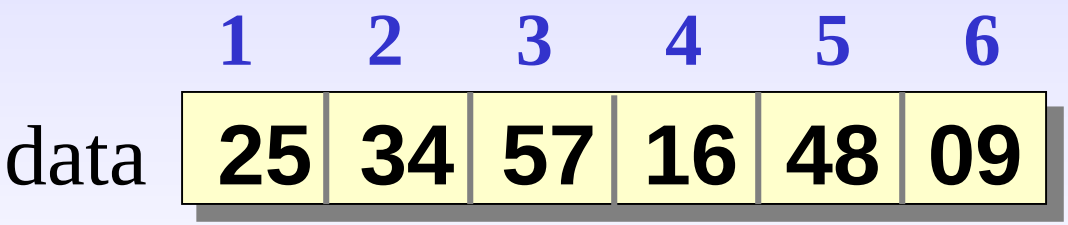
第二章　线 性 表

- 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
  - 2.3.1　线性链表
  - 2.3.2　循环链表
  - 2.3.3　双向链表
- 2.4 一元多项式的表示及相加

2.2 线性表的顺序表示和实现

1、顺序表的定义

把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。



用物理位置来表示逻辑结构。

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

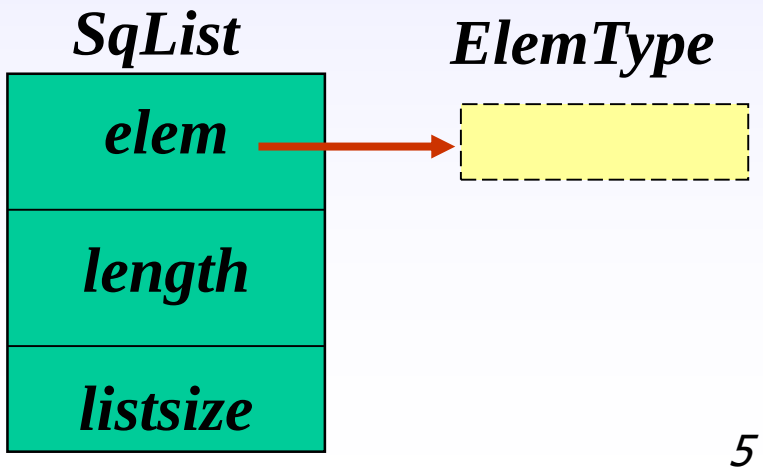
$LOC(a_i)=LOC(a_1)+(i-1)*l$

特点：
随机存取的存储结构。只要确定了存储线性表的起始位置，线性表中的任一数据元素可随机存取。

2.2 线性表的顺序表示和实现

2、线性表的动态分配顺序存储结构（动态一维数组）

```
#define LIST_INIT_SIZE   100   //初始分配量
#define LISTINCREMENT   10  //分配增量
typedef struct{
   ElemType *elem;  //存储空间基址
   int  length;    //当前长度
   int  listsize;   //当前存储容量
}SqList
```



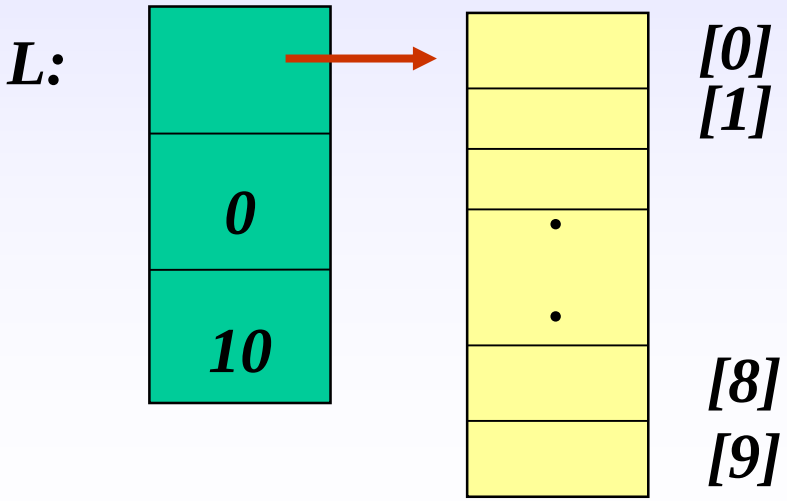
2.2 线性表的顺序表示和实现

3、顺序表的操作

(1) 初始化操作

算法思想：构造一个空表。

设置表的起始位置、表长及可用空间。



2.2 线性表的顺序表示和实现

算法：

```
Status InitList_Sq(SqList &L){  //构造一个空的线性表
        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 线性表的顺序表示和实现

（2）在线性表中指定位置前插入一个元素

插入第i个元素时，线性表的逻辑结构由 $(a_1,\ldots,a_{i-1},a_i,\ldots,a_n)$ 改变为 $(a_1,\ldots,a_{i-1},e,a_i,\ldots,a_n)$

算法思想：

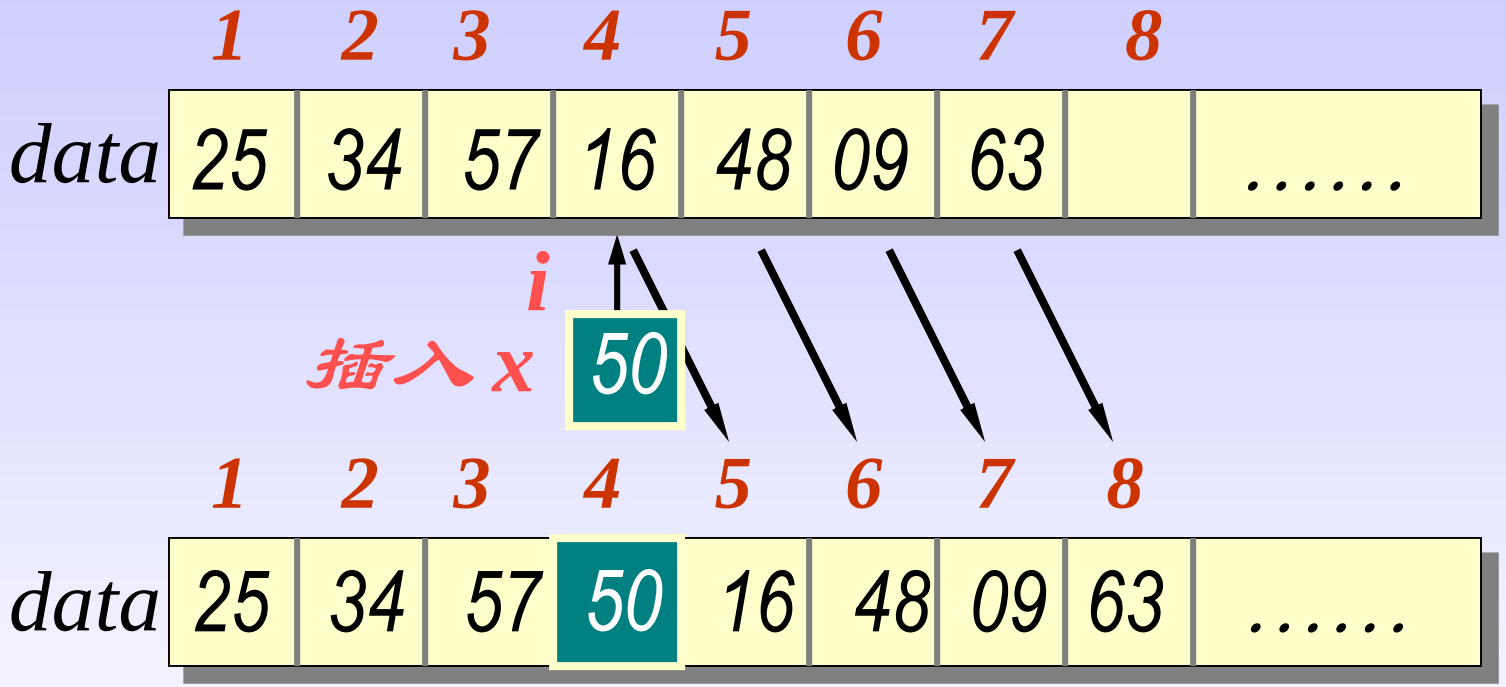
1）检查i值是否超出所允许的范围（$1\le i\le n+1$），若超出，则进行“超出范围”错误处理；

2）将线性表的第i个元素和它后面的所有元素均向后移动一个位置；

3）将新元素写入到空出的第i个位置上；

4）使线性表的长度增1。

2.2 线性表的顺序表示和实现



平均移动次数：

$$AMN = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{1}{n+1}(n+\cdots+1+0)$$

$$= \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

2.2 线性表的顺序表示和实现

```
Status ListInsert_Sq(SqList &L, int i, ElemType e)
{  if (i < 1 || i > L.length+1) return ERROR;    // 插入位置不合法
   if (L.length >= L.listsize) {     // 当前存储空间已满，增加分配
      newbase = (ElemType *)realloc
        (L.elem,(L.listsize+LISTINCREMENT)*sizeof (ElemType));
     if (!newbase) exit(OVERFLOW); // 存储分配失败
     L.elem = newbase; // 新基址
     L.listsize += LISTINCREMENT; // 增加存储容量
   }
   q = &(L.elem[i-1]); // q指示插入位置
   for (p = &(L.elem[L.length-1]); p>=q;--p)
       *(p+1)=*p;
   *q=e;
   ++ L.length;
   return  OK;  }
```

2.2 线性表的顺序表示和实现

（3）在线性表中删除第i个元素

删除元素时，线性表的逻辑结构由$(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$改变为$(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$

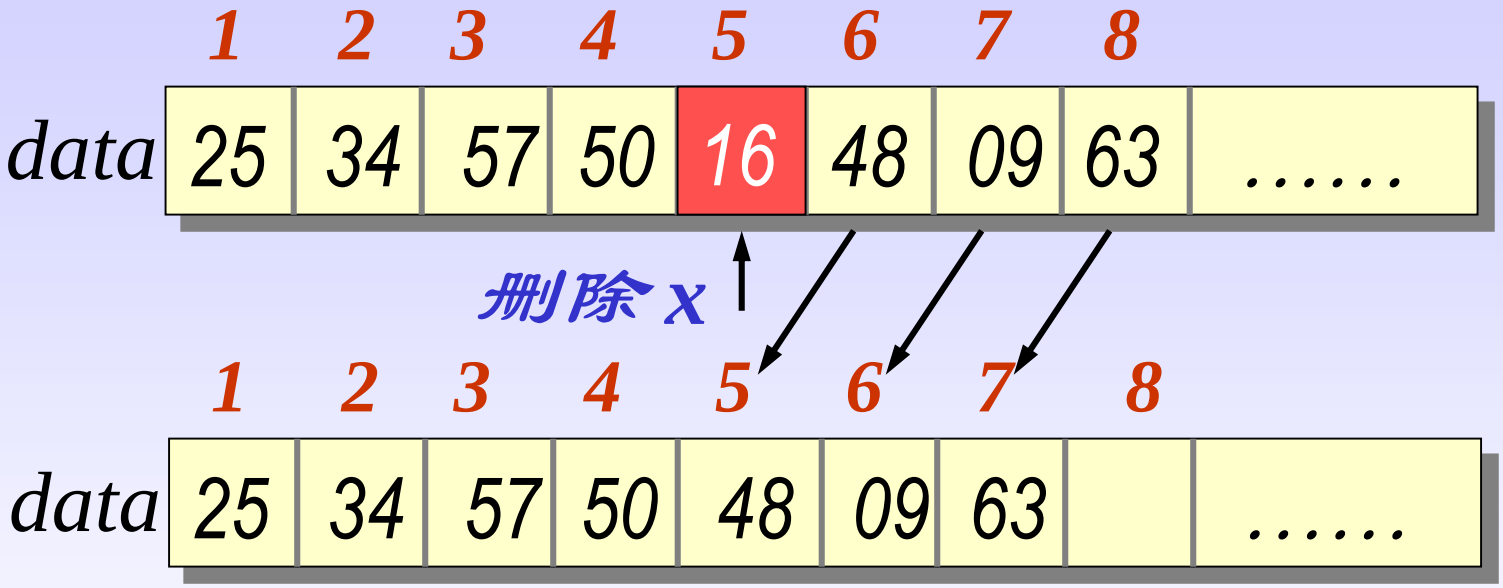
算法思想：

1）检查i值是否超出所允许的范围（$1 \le i \le n$），若超出，则进行“超出范围”错误处理；

2）将线性表的第i个元素后面的所有元素均向前移动一个位置；

3）使线性表的长度减1。

2.2 线性表的顺序表示和实现



$$\mathrm{AMN} = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

2.2 线性表的顺序表示和实现

```
Status ListDelete_Sq(SqList &L, int i, ElemType &e)
{   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;
}
```

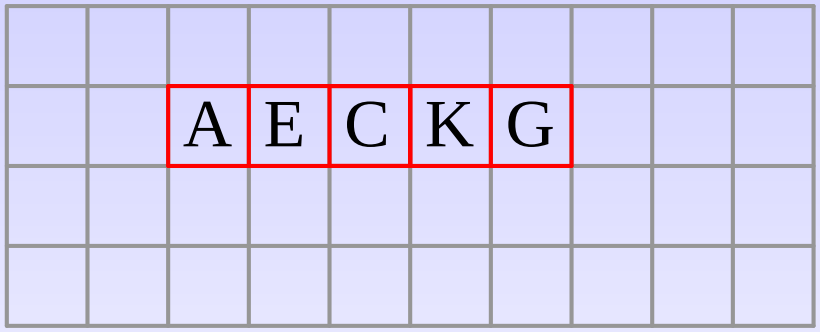
2.2 线性表的顺序表示和实现

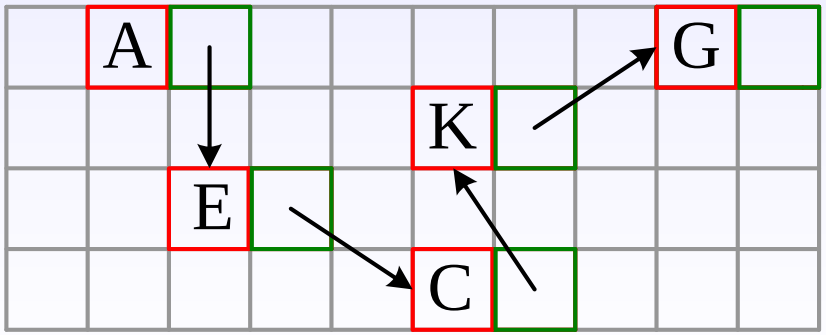
```
Status ListDelete_Sq(SqList &L, int i, ElemType &e)
{   if ((i < 1) || (i > L.length)) return ERROR;    // 删除位置不合法
    p = &(L.elem[i-1]);       // p为被删除元素的位置
    e = *p;                   // 被删除元素的值赋给e
    q = L.elem+L.length-1; // 表尾元素的地址
    for (++p; p <= q; ++p)
        *(p-1) = *p;          // 被删除元素之后的元素左移
     - - L.length;
     return  OK;
}
```

2.2 线性表的顺序表示和实现

- 顺序存储结构的优缺点
  - 优点
    - 逻辑相邻，物理相邻
    - 可随机（直接）存取任一元素
    - 存储空间使用紧凑
  - 缺点
    - 插入、删除操作需要移动大量的元素
    - 预先分配空间需按最大空间分配，利用不充分

2.3 线性表的链式表示与实现

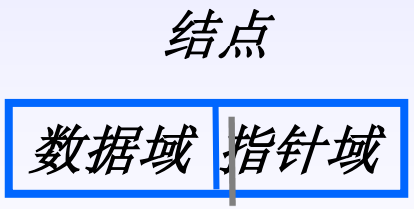




2.3 线性表的链式表示与实现

特点：

- 用一组任意的存储单元存储线性表的数据元素
- 利用指针实现了用不相邻的存储单元存放逻辑上相邻的元素
- 每个数据元素a_i，除存储本身信息外，还需存储其直接后继的信息
- 结点

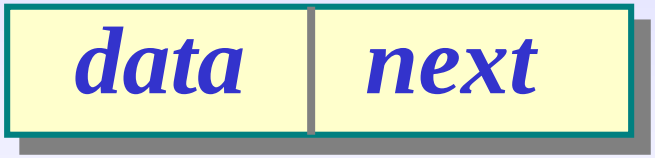


  - 数据域：元素本身信息
  - 指针域：指示直接后继的存储位置

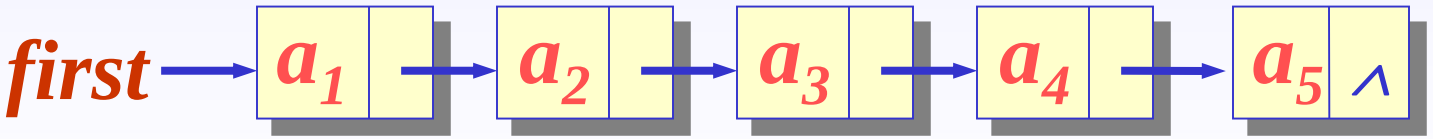
2.3 线性表的链式表示与实现

2.3.1 单链表（或线性链表）

1. 定义：结点中只含一个指针域的链表
2. 特征：
   - 每个元素(表项)由结点(*Node*)构成。



   - 线性结构



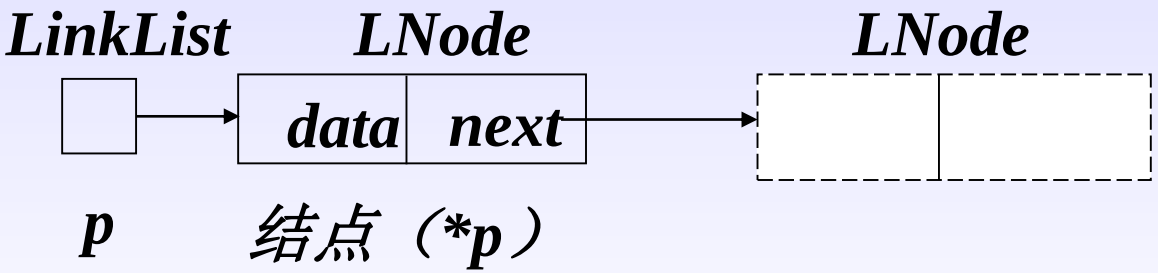
   - 结点可以不连续存储
   - 表可扩充

2.3 线性表的链式表示与实现

4. 单链表存储结构的实现

——用C语言中的“结构指针”来描述

```
typedef struct LNode {
    ElemType data;
    struct LNode  *next;
} LNode ，  *LinkList;
```



注意结点p与结点a_i的区别

(*p) 表示p所指向的结点

(*p).data⇔p->data 表示p指向结点的数据域

(*p).next⇔p->next 表示p指向结点的指针域

2.3 线性表的链式表示与实现

单链表特点：

- 它是一种动态结构，整个存储空间为多个链表共用
- 不需预先分配空间
- 指针占用额外存储空间
- 不能随机存取，查找速度慢

生成一个LNode型新结点：

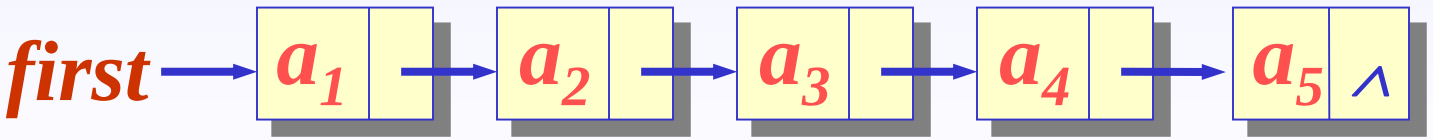
```
p=(LinkList)malloc(sizeof(LNode));
```

系统回收p结点： ***free(p)***

2.3 线性表的链式表示与实现

插入

- 第一种情况：在第一个结点前插入
- 第二种情况：在链表中间插入
- 第三种情况：在链表末尾插入

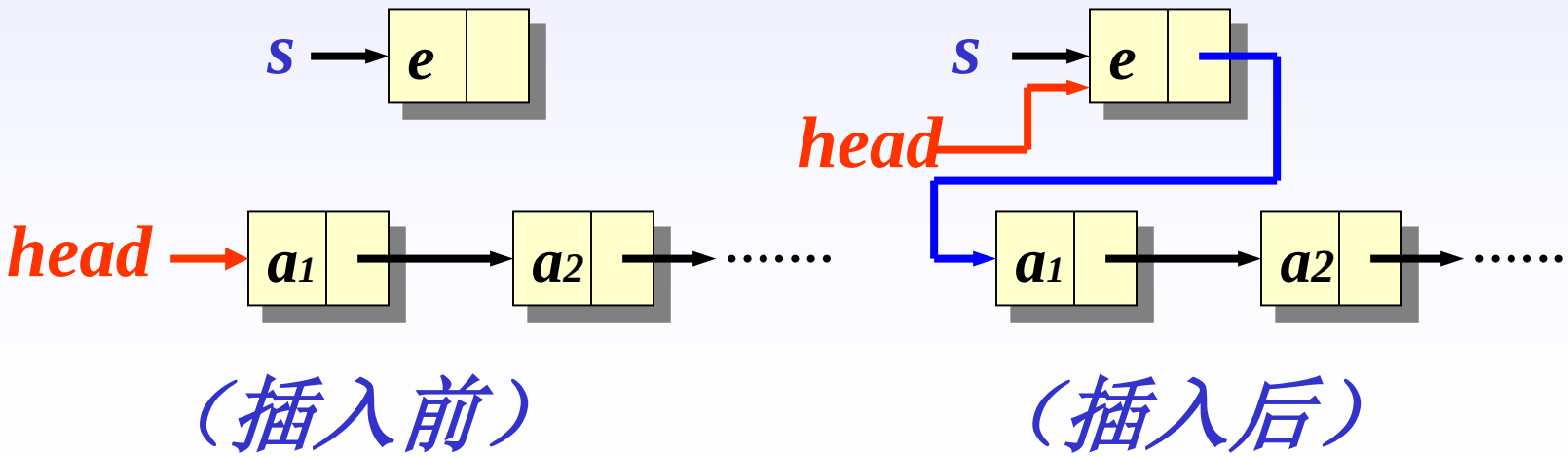


2.3 线性表的链式表示与实现

插入

- 第一种情况：在第一个结点前插入

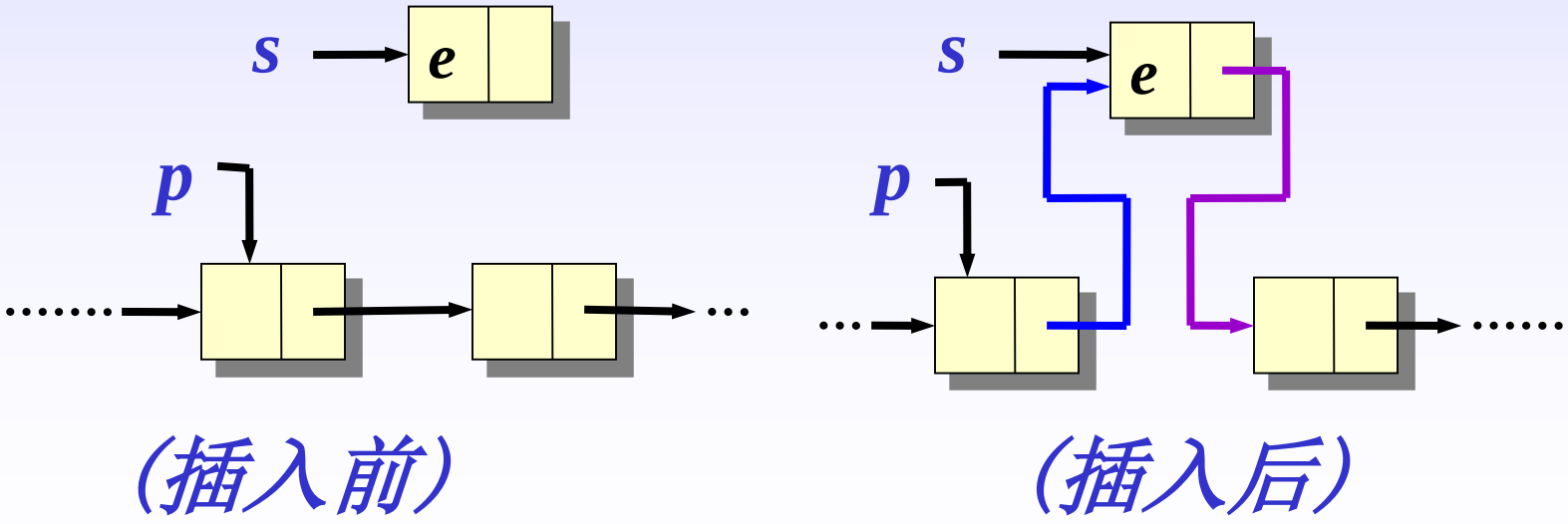
```
s->next = head ;
head = s;
```



2.3 线性表的链式表示与实现

- 第二种情况：在链表中间插入

```
s->next = p->next;
p->next = s;
```

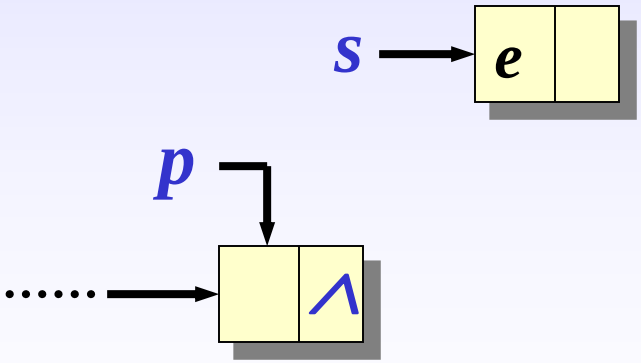


(插入前)　　(插入后)

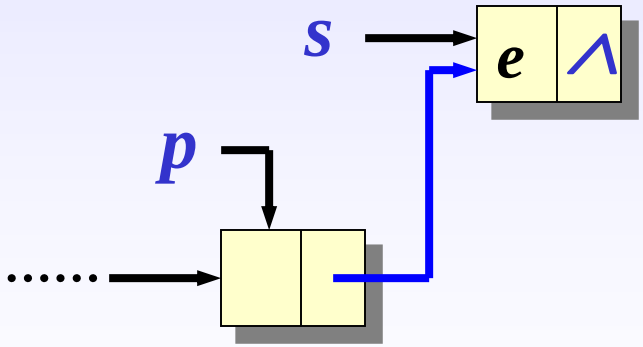
2.3 线性表的链式表示与实现

- 第三种情况：在链表末尾插入

```
s->next= p->next;
p->next = s;
```



（插入前）

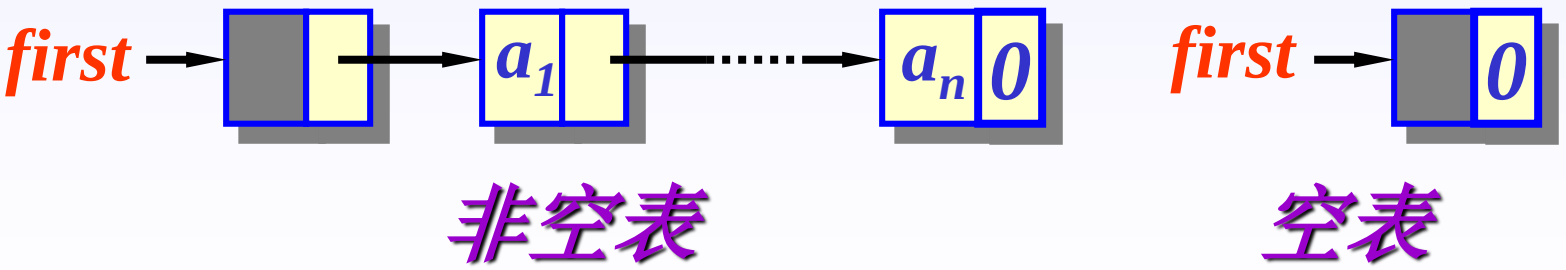


（插入后）

2.3 线性表的链式表示与实现

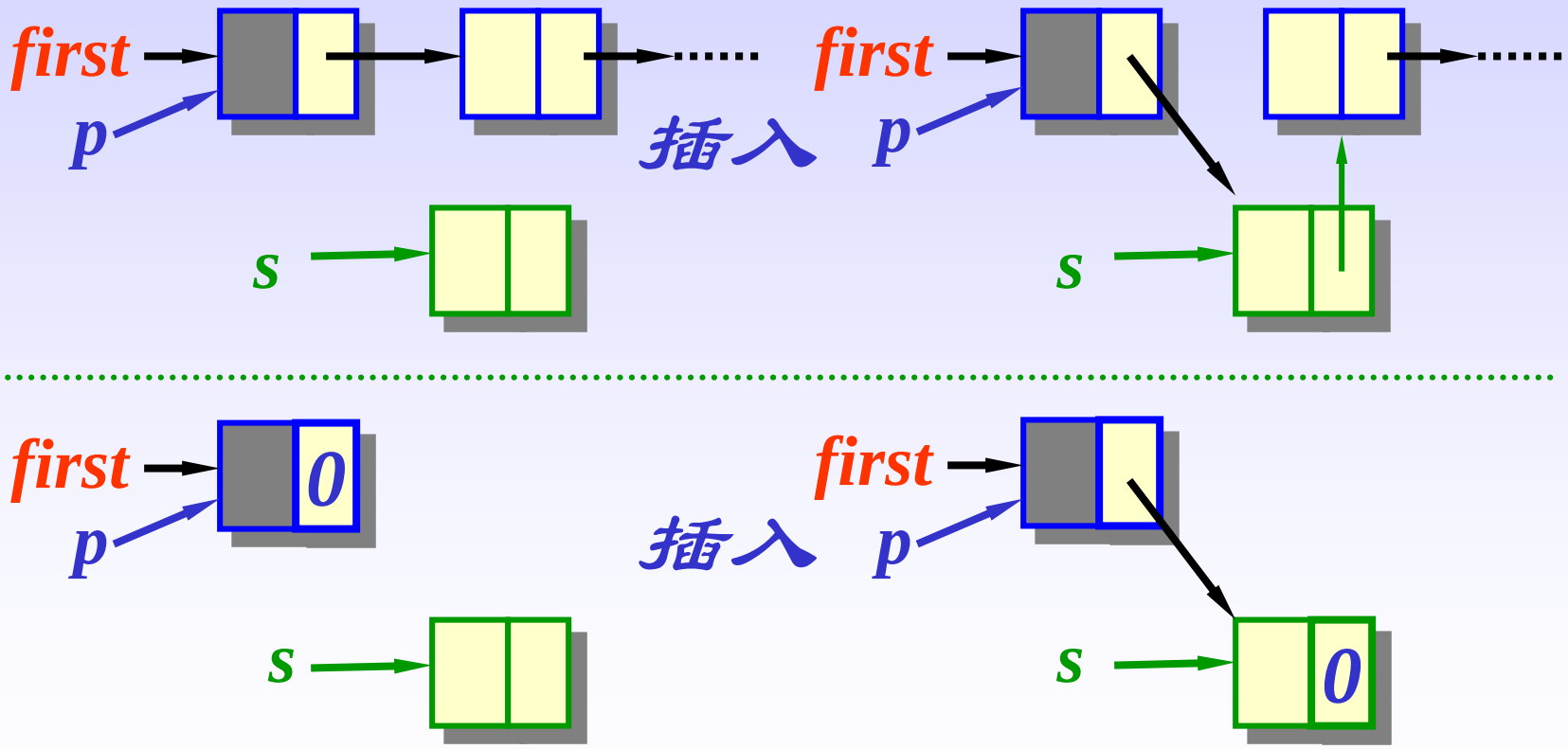
5. 带头结点的单链表

- 头结点位于表的最前端，本身不带数据，仅标志表头。
- 设置头结点的目的是统一空表与非空表的操作，简化链表操作的实现。



2.3 线性表的链式表示与实现

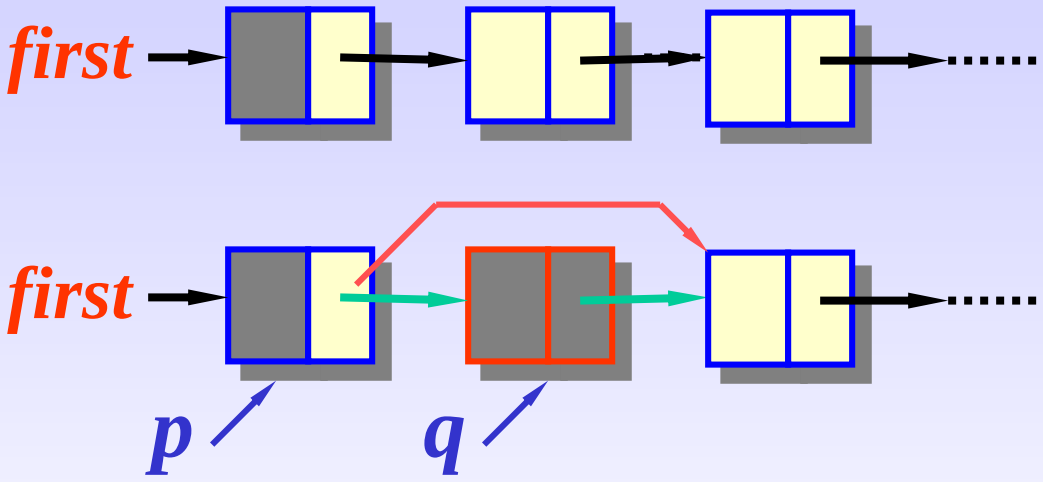
(1)在带头结点的单链表第一个结点前插入新结点



s->next = p->next; p->next = s;

2.3 线性表的链式表示与实现

(2)从带头结点的单链表中删除第一个结点




```
q = p->next;
p->next = q->next;
free(q);
```

2.3 线性表的链式表示与实现

6. 单链表的若干算法及操作

(1) 根据已有数据生成一个链表

CreateList_L( )

(2) 插入

ListInsert_L ( )

(3) 删除

ListDelete_L ( )

(4) 归并两个链表

MergeList_L ( )

2.3 线性表的链式表示与实现

(1) 根据已有数据生成一个链表

设线性表n个元素已存放在数组a中，建立一个单链表，h为头指针。

算法思路：

（1）创建头结点

（2）逆序创建结点n，修改结点指针，再创建结点n-1,……直到结点1

2.3 线性表的链式表示与实现

```
void CreateList_L ( LinkList &L, int n ) {
   // 逆位序输入 n 个数据元素的值，建立带头结点的单链表 L
   L = ( LinkList ) malloc ( sizeof ( LNode ) );
   L->next = NULL; // 先建立一个带头结点的空链表
   for ( i = n; i > 0; --i ) {
         p = ( LinkList ) malloc ( sizeof ( LNode ) ); // 生成新结点
         scanf ( &p->data ); // 按照逆位序输入数据元素值
         p->next = L->next; // 将新结点插入到单链表的头
         L->next = p; // 修改单链表头结点的指针域
   } // for 结束
} CreatList_L
```

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

2.3 线性表的链式表示与实现

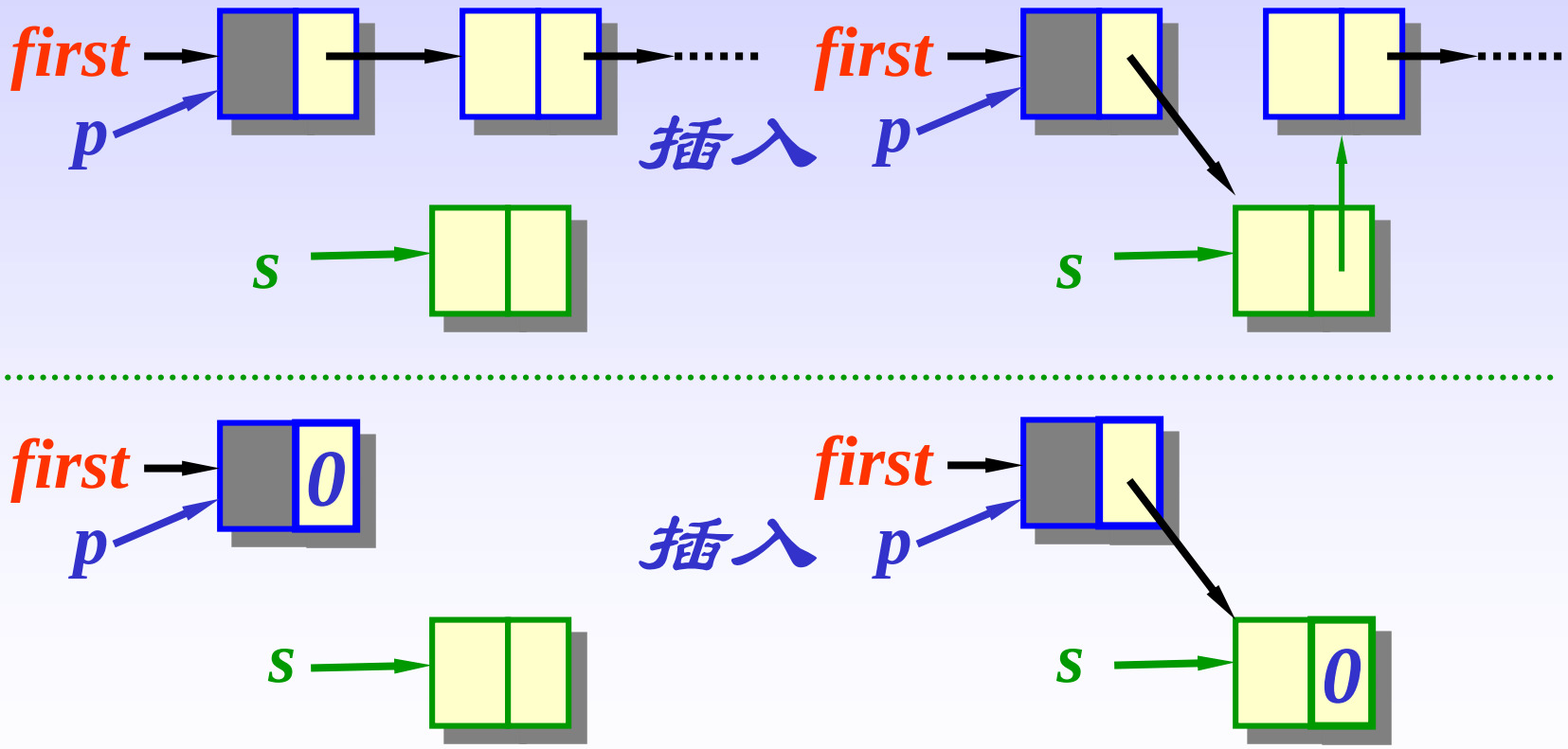
(2) 插入

```
Status ListInsert_L ( LinkList &L, int i, ElemType e ) {
// 在带头结点的单链线性表L中第 i 个位置之前插入元素 e
  p = L; j = 0;
  while ( p && j<i-1 ) { p = p->next; ++j; } // 寻找第 i-1 个结点
  if ( ! p || j >i-1 ) return ERROR; // i 小于 1 或 i 大于表长
  s = ( LinkList ) malloc ( sizeof ( LNode ) ); // 生成新结点
  s->data = e; // 使新结点数据域的值为 e
  s->next = p->next; // 将新结点插入到单链表 L 中
  p->next = s; // 修改第 i-1 个结点指针
  return OK;
} // LinkInsert_L
```

$T(n)=O(n)$

2.3 线性表的链式表示与实现

(1)在带头结点的单链表第一个结点前插入新结点

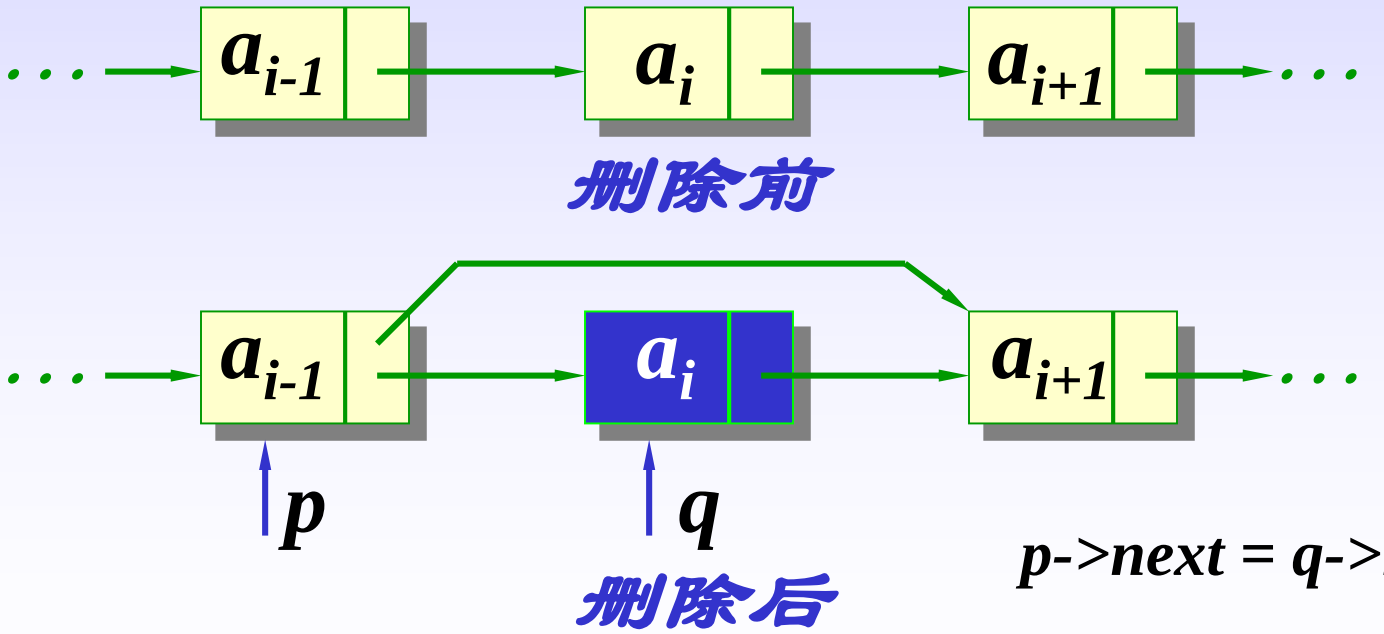


s->next = p->next; p->next = s;

2.3 线性表的链式表示与实现

(3) 删除

- 第一种情况: 删除表中第一个元素
- 第二种情况: 删除表中或表尾元素



在单链表中删除含a_i的结点

2.3 线性表的链式表示与实现

```
Status ListDelete_L ( LinkList &L, int i, ElemType &e ) {
// 在带头结点的单链线性表 L 中，删除第 i 个元素，并由 e 返回其值
  p = L; j = 0;
  while ( p->next && j<i-1 ) { // 寻找第 i-1 个结点
    p = p->next; ++j;
  }
  if ( ! p || j >i-1 ) return ERROR; // 删除位置不合理
  q = p->next;   // 用指针 q 指向被删除结点
  p->next = q->next;   // 删除第 i 个结点
  e = q->data;    // 取出第 i 个结点数据域值
  free (q);     // 释放第 i 个结点
  return OK;
} // LinkDelete_L
```

$T(n)=O(n)$

2.3 线性表的链式表示与实现

(4) 归并两个链表

```
void MergeList_L ( LinkList &La, LinkList &Lb, LinkList &Lc, )
{// 已知单链表 La 和 Lb 的元素按非递减排列，
    // 归并 La 和 Lb 得到新的单链表 Lc ， Lc 的元素也按非递减排列
    pa = La->next; pb = Lb->next;
    Lc = pc = La; // 用 La 的头结点作为 Lc 的头结点
    while ( pa && pb ) {
        if ( pa->data <= pb->data ) // 如果 pa->data≤pb->data
        {       pc->next = pa;   pc = pa;   pa = pa->next;     }
        else // 如果 pa->data > pb->data
        {       pc->next = pb;    pc = pb;   pb = pb->next;    }
    }
    pc->next = pa ? pa : pb; // 插入剩余段
    free (Lb); // 释放 Lb 的头结点
} // MergeList_L
```