数据结构

（C语言版）（第2版）

线性表

线性表单链表表示与实现

主讲教师：汪红松

教 学 内 容 Contents

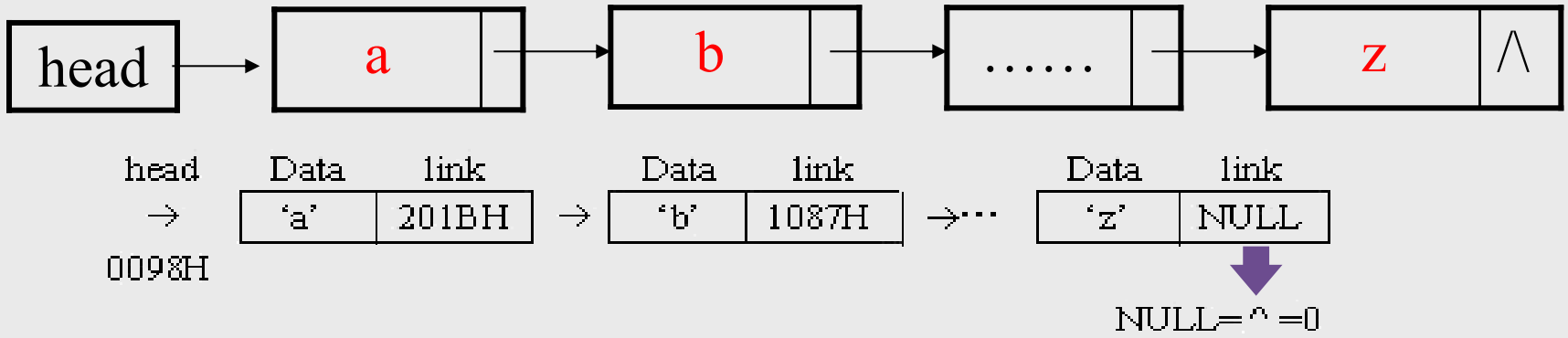
链式存储结构

结点在内存中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻。

例　画出26 个英文字母表的链式存储结构

逻辑结构：（ a, b, … ,y, z ）

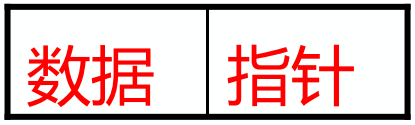
链式存储结构：



各结点由两个域组成：

数据域：存储元素数值数据；

指针域：存储直接后继结点的存储位置。

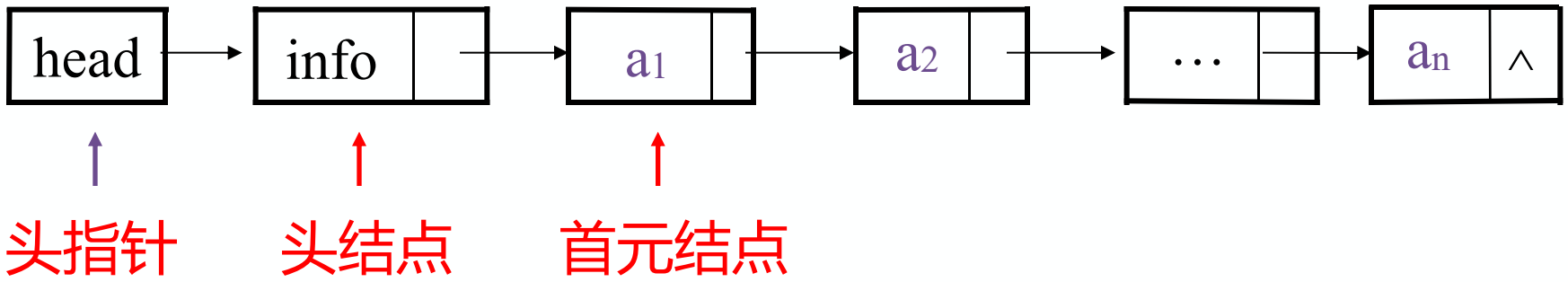


结点只有一个指针域的链表，称为单链表或线性链表。

一、链表

1.与链式存储有关的术语

头指针、头结点和首元结点



头指针是指向链表中第一个结点的指针。

头结点是在链表的首元结点之前附设的一个结点；数据域内只放空表标志和表长等信息。

首元结点是指链表中存储第一个数据元素a_1的结点。

一、链表

2.链表（链式存储结构）的特点

(1) 结点在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻。

(2) 访问时只能通过头指针进入链表，并通过每个结点的指针域向后扫描其余结点，所以寻找第一个结点和最后一个结点所花费的时间不等。

这种存取元素的方法被称为链式存取法。

3.链表的优缺点

（1）优点

- 数据元素的个数可以自由扩充；
- 插入、删除等操作不必移动数据，只需修改链接指针，修改效率较高。

3.链表的优缺点

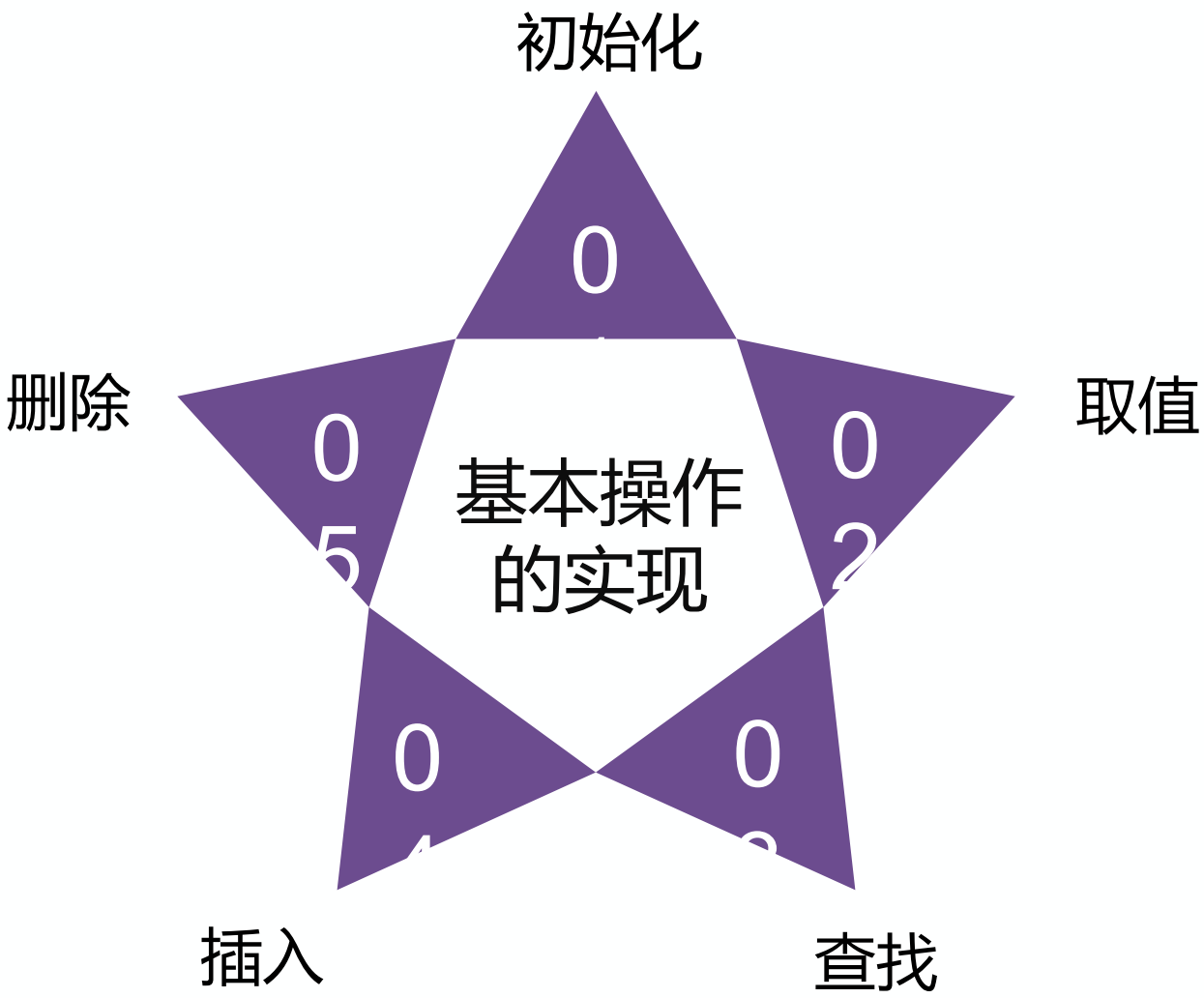
（2）缺点

存取效率不高，必须采用顺序存取，即存取数据元素时，只能按链表的顺序进行访问（顺藤摸瓜）。

二、单链表

1.单链表的存储结构定义

```
typedef struct Lnode
{
    ElemType   data;       //数据域
    struct LNode  *next;   //指针域
}LNode,  * LinkList;
// *LinkList  为  Lnode  类型的指针
```

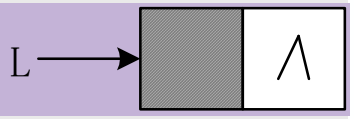

初始化
取值
查找
插入
删除
基本操作的实现

（1）初始化(构造一个空表）

【算法步骤】

① 生成新结点作头结点，用头指针L指向头结点。

② 头结点的指针域置空。

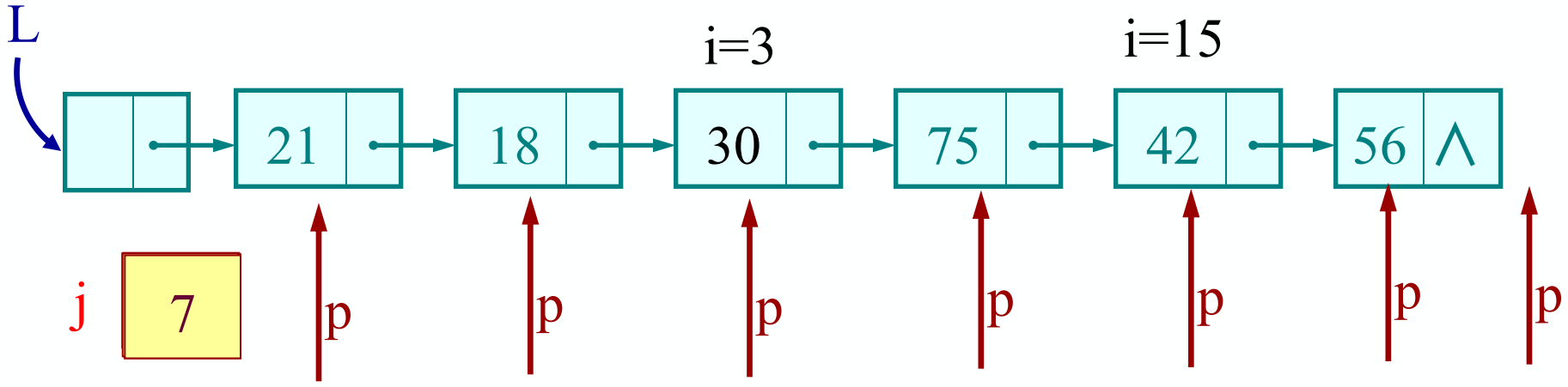


【算法描述】

```
Status InitList_L(LinkList &L){
   L=new LNode;
   L->next=NULL;
   return OK;
}
```

线性表的重要基本操作

例：分别取出表中i=3和i=15的元素



【算法步骤】

✓ 从第1个结点（L->next）顺链扫描，用指针p指向当前扫描到的结点，p初值p = L->next。

✓ *j*做计数器，累计当前扫描过的结点数，*j*初值为1。

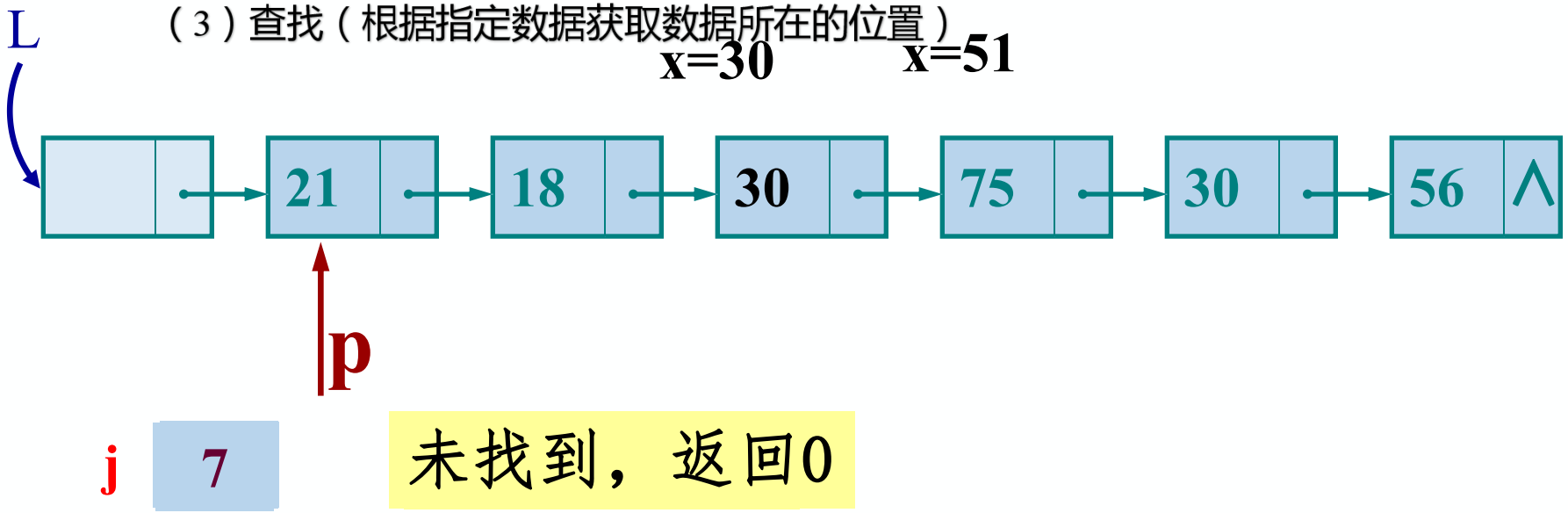
✓ 当p指向扫描到的下一结点时，计数器*j*加1。

✓ 当*j* = *i*时，p所指的结点就是要找的第*i*个结点。

（2）取值（根据位置i获取相应位置数据元素的内容）

```
//获取线性表L中的某个数据元素的内容
Status GetElem_L(LinkList L,int i,ElemType &e){
    p=L->next;j=1; //初始化
     while(p&&j<i){     //向后扫描，直到p指向第i个元素或p为空
       p=p->next; ++j;
     }
     if(!p || j>i)return ERROR; //第i个元素不存在
     e=p->data; //取第i个元素
     return OK;
}//GetElem_L
```

（3）查找（根据指定数据获取数据所在的位置）



- ✓从第一个结点起，依次和e相比较；
- ✓如果找到一个其值与e相等的数据元素，则返回其在链表中的“位置”或地址；
- ✓如果查遍整个链表都没有找到其值和e相等的元素，则返回0或“NULL”。

线性表的重要基本操作

【算法描述】

```
//在线性表L中查找值为e的数据元素
LNode *LocateELem_L (LinkList L，Elemtype e) {
 //返回L中值为e的数据元素的地址，查找失败返回NULL
  p=L->next;
  while(p &&p->data!=e)
        p=p->next;
  return p;
}
```

线性表的重要基本操作

【算法描述】

```
//在线性表L中查找值为e的数据元素
int LocateELem_L (LinkList L，Elemtype e) {
 //返回L中值为e的数据元素的位置序号，查找失败返回0
  p=L->next; j=1;
  while(p &&p->data!=e)
        {p=p->next;  j++;}
  if(p) return j;
  else return 0;
}
```

线性表的重要基本操作

将值为x的新结点插入到表的第i个结点的位置上，即插入到a_{i-1}与a_i之间。

【算法步骤】

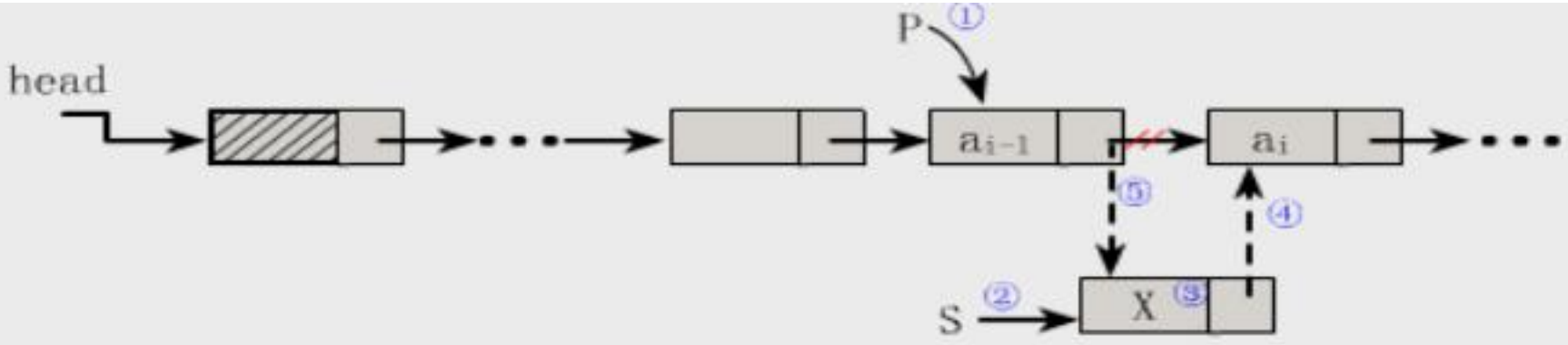
01 OPTION 找到a_{i-1}存储位置p

02 OPTION 生成一个新结点*s；

03 OPTION 将新结点*s的数据域置为x；

04 OPTION 新结点*s的指针域指向结点a_i；

05 OPTION 令结点*p的指针域指向新结点*s。



在单链表上插入结点示意图

线性表的重要基本操作

【算法描述】

```
//在L中第i个元素之前插入数据元素e
Status ListInsert_L(LinkList &L,int i,ElemType e){
    p=L;j=0;
    while(p&&j<i-1){p=p->next;++j;}        //①寻找第i-1个结点
    if(!p||j>i-1)return ERROR;             //i大于表长 + 1或者小于1
    s=new LNode;                           //②生成新结点s
    s->data=e;                             //③将结点s的数据域置为e
    s->next=p->next;                       //④将结点s插入L中
    p->next=s;                             //⑤
    return OK;
}//ListInsert_L
```

（4）删除（删除第 i 个结点）

① 将表的第i个结点删去；

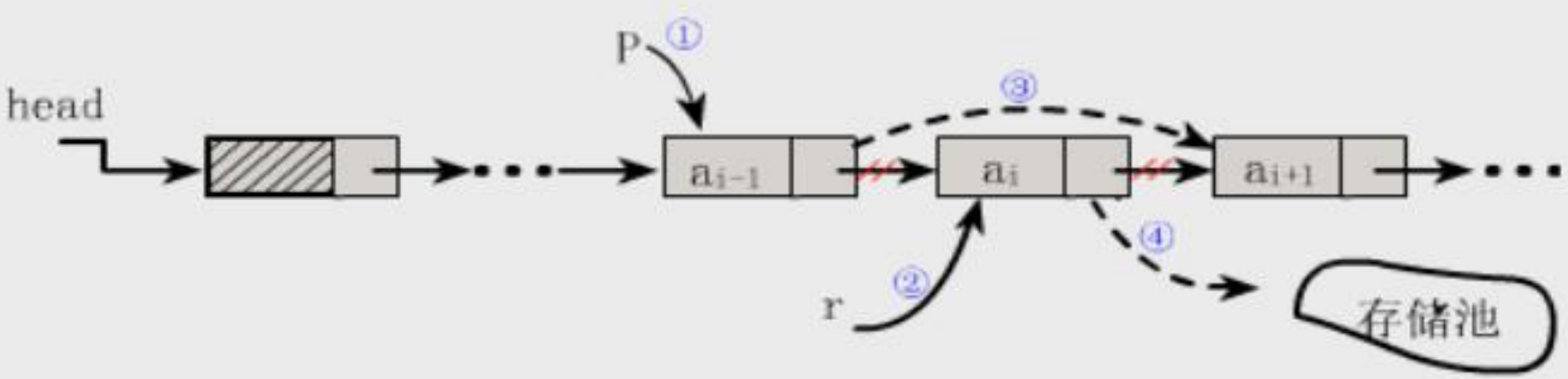
② 步骤：

1）找到a_{i-1}存储位置p；

2）保存要删除的结点的值；

3）令p->next指向a_i的直接后继结点；

4）释放结点a_i的空间。



在单链表上删除结点示意图

（4）删除（删除第 i 个结点）

【算法描述】

```
//将线性表L中第i个数据元素删除
 Status ListDelete_L(LinkList &L,int i,ElemType &e){
   p=L;j=0;
   while(p->next &&j<i-1){                    //①寻找第i个结点，并令p指向其前驱
      p=p->next; ++j;
   }
   if(!(p->next)||j>i-1) return ERROR; //删除位置不合理
   q=p->next;                                 //②临时保存被删结点的地址以备释放
   p->next=q->next;                           //③改变删除结点前驱结点的指针域
   e=q->data;                                 //保存删除结点的数据域
   delete q;                                  //④释放删除结点的空间
 return OK;
}//ListDelete_L
```

三、链表的运算时间效率分析

1. 查找：因线性链表只能顺序存取，即在查找时要从头指针找起，查找的时间复杂度为 O(n)。

2. 插入和删除：因线性链表不需要移动元素，只要修改指针，一般情况下时间复杂度为 O(1)。

如果要在单链表中进行前插或删除操作，由于要从头查找前驱结点，所耗**时间复杂度为 O(n)** 。