数据结构与算法
Data Structure and Algorithm

第2章　线性表

目　录

2.1.1线性表的定义

1. 线性表的定义

线性表（linear_List）是n（n≥0）个数据元素a_1，a_2，…a_n组成的有限序列。其中n 称为数据元素的个数或线性表的长度，当n=0时称为空表，n>0时称为非空表。

通常将非空的线性表记为（a_1，a_2，…,a_n），其中的数据元素a_i（1≤i≤n）是一个抽象的符号，其具体含义在不同情况下是不同的，即它的数据类型可以根据具体情况而定，本书中，我们将它的类型设定为ElemSet，表示某一种具体的已知数据类型。

- 例1、26个英文字母组成的字母表
- （A，B，C、…、Z）
- 例2、某校从1978年到1983年各种型号的计算机拥有量的变化情况。
- （6，17，28，50，92，188）
- 例3、一副扑克的点数
- （2，3，4，…，J，Q，K，A）
- 例4、学生健康情况登记表如下：

姓　名	学　　号	性　别	年龄	健康情况
王小林	790631	男	18	健康
陈　红	790632	女	20	一般
刘建平	790633	男	21	健康
张立立	790634	男	17	神经衰弱
……	……	……	……	……

2. 线性表的特征

从线性表的定义可以看出线结构的特征：

（1）有且仅有一个开始结点(表头结点)a_1，它没有直接前驱，只有一个直接后继;

（2）有且仅有一个终端结点(表尾结点)a_n，它没有直接后继，只有一个直接前驱;

（3）其它结点都有一个直接前驱和直接后继；

（4）元素之间为一对一的线性关系。

线性表是一种典型的线性结构。

2.1.2 线性表的抽象数据类型描述 P19

说明：

1. 某数据结构上的基本运算，不是它的全部运算，而是一些常用的基本的运算，而每一个基本运算在实现时也可能根据不同的存储结构派生出一系列相关的运算来。比如线性表的查找在链式存储结构中还会有按序号查找; 再如插入运算，也可能是将新元素 x 插入到适当位置上等等，不可能也没有必要全部定义出它的运算集，读者掌握了某一数据结构上的基本运算后，其它的运算可以通过基本运算来实现，也可以直接去实现。

2. 此处定义的线性表 L 仅仅是一个抽象在逻辑结构层次的线性表，尚未涉及到它的存储结构，因此每个操作在逻辑结构层次上尚不能用具体的某种程序语言写出具体的算法，而算法的实现只有在存储结构确立之后。

写算法三步曲：

1. 根据功能要求举最直观的例子，并演示其整个操作过程；
2. 把演示过程中的每一步转为语句；
3. 把语句进行整合，加上算法的头尾，形成算法。

算法说明：

算法2. 1

- 例2-1 利用两个线性表LA和LB分别表示两个集合A和B，现要求一个新的集合A=A∪B。

```
void union(List &La，List Lb) {
    //将所有在线性表Lb中但不在La中的数据元素插入到La中
    La_len=ListLength(La);
    Lb_len=ListLength(Lb);
    for(i=1;i<=lb_len;i++) {
        getelem(lb，i，&e);
        if(!LocateElem(La，e，equal)) ListInsert(La，++la_len，e)
    }
}//union
```

算法2.2

- 例2-2 已知线性表LA和线性表LB中的数据元素按值非递减有序排列，现要求将LA和LB归并为一个新的线性表LC，且LC中的元素仍按值非递减有序排列。

此问题的算法如下：

```
void  MergeList (List La，List Lb，List &Lc){
    //已知线性表La和Lb中的数据元素按值非递减排列。
    //归并La和Lb得到新的线性Lc, Lc的数据元素也按值非递减排列
    InitList (Lc);
    i=j=1;k=0;
    La_Len = ListLength(La);  lb-Len = ListLength(Lb);
    while((i<=La_Len) && (j<=Lb_Len)){
        GetElem(La，i，&ai); GetElem(Lb，j，&bj);
        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，bi);
    }
}// mergeList
```

2.2线性表的顺序存储结构

2.2.1 顺序表结构

1. 定义

线性表的顺序存储结构，也称为顺序表。其存储方式为：在内存中开辟一片连续存储空间，但该连续存储空间的大小要大于或等于顺序表的长度，然后让线性表中第一个元素存放在连续存储空间第一个位置，第二个元素紧跟着第一个之后，其余依此类推。

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。用这种方法存储的线性表简称顺序表。

假设线性表的每个元素需占用d个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储位置。则线性表中第i+1个数据元素的存储位置LOC(a_{i+1})和第i个数据元素的存储位置LOC(a_i)之间满足下列关系：

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

线性表的第i个数据元素ai的存储位置为：

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

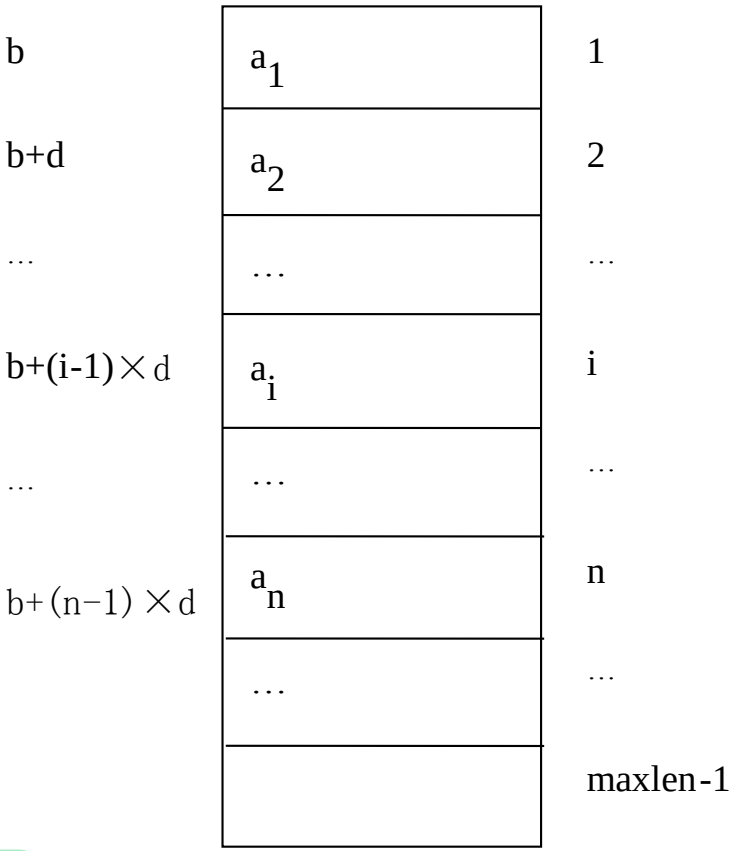


图 2 -2 顺序存储结构示意图

2. 线性表的动态分配顺序存储结构

- 由于C语言中的一维数组也是采用顺序存储表示，故可以用数组类型来描述顺序表。又因为线性表的长度可变，且所需最大存储空间可随问题而不同，所以我们用动态分配的一维数组结构类型来定义顺序表类型。

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

typedef  struct{
    ElemType  *elem;    //存储空间的基址
    int   length;       //当前的长度
    int   listsize;     //当前分配的存储容量
} Sqlist;
```

注意：C语言中数组的下标从“0”开始，则线性表的第i个元数表示为L.elem[i-1]

- 算法 2.3 ，构造一个空的线性表 P23

- 注意：

```
L.elem = (ElemType * ) malloc (LIST_INIT_SIZE * sizeof(ElemType));
if(! L.elem) exit(OVERFLOW);          //初始化


newbase =  (ElemType * ) realloc (L.elem ,(L.listsize +
  L.listincrement)* sizeof( ElemType ));
if(! newbase) exit(OVERFLOW);
L.elem= newbase;
L.listsize+= L.LISTINCREMENT;     //增加大小为listincrement存储空间
```

2.2.2 顺序表运算

1．求顺序表的长度

```
int length(Sqlist &L){
  return L.length;
}//length
```

该算法的语句频度为1，时间复杂度为0(1)。

2．插入运算

线性表的插入运算是指在表的第I(1≦i≦n+1个位置上，插入一个新结点x，使长度为n的线性表

(a1，…a i-1，ai，…，an)

变成长度为n+1的线性表

(a1，…a i-1，x，ai，…，an)

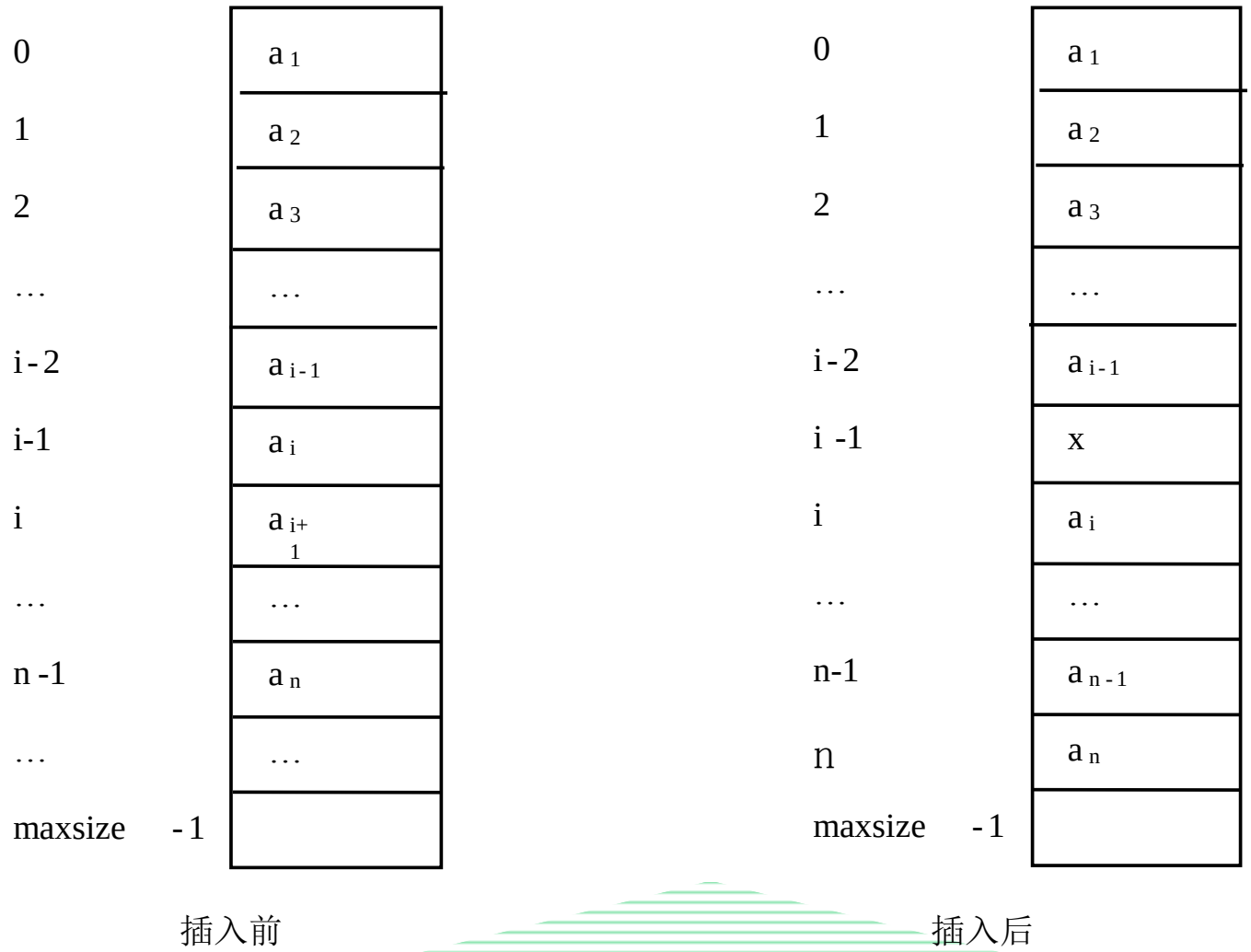


图 2-3 顺序表中插入元素前后状态

算法为：

```
Status ListInsert_Sq (Sqlist  &L，ElemType e，int I {
    //在顺序线性表L中第i个位置之前插入新的元数e
    //i的合法值为i<=ListLength_Sq(L)+1
    if ( i<1 || i >L.length+1)    return ERROR;
    if(L.length>=ListSize){//当前存储空间已满，增加分配
       newbase = （ElemType * ）realloc (L.elem ,(L.listsize + L.listincrement)*
                        sizeof( ElemType ));
       if(! newbase) exit(OVERFLOW);
       L.elem= newbase;
       L.listsize+= L.LISTINCREMENT;
    }
    q=&(L.elem[i-1]);  //q为插入位置
    for(p=&(L.elem[L.length-1]); p>=q;- - p)   * (p+1)= * p;
                        //插入位置之后的元数右移
    * q= e;             // 插入 e
    ++L.length;          //表长增1
    return  OK;
}// ListInsert_Sq
```

3．删除运算

线性表的删除运算是指将表的第i(1≦i≦n)结点删除，使长度为n的线性表：

(a1，…a i-1，ai，a i+1…，an)

变成长度为n-1的线性表

(a1，…a i-1，a i+1，…，an)

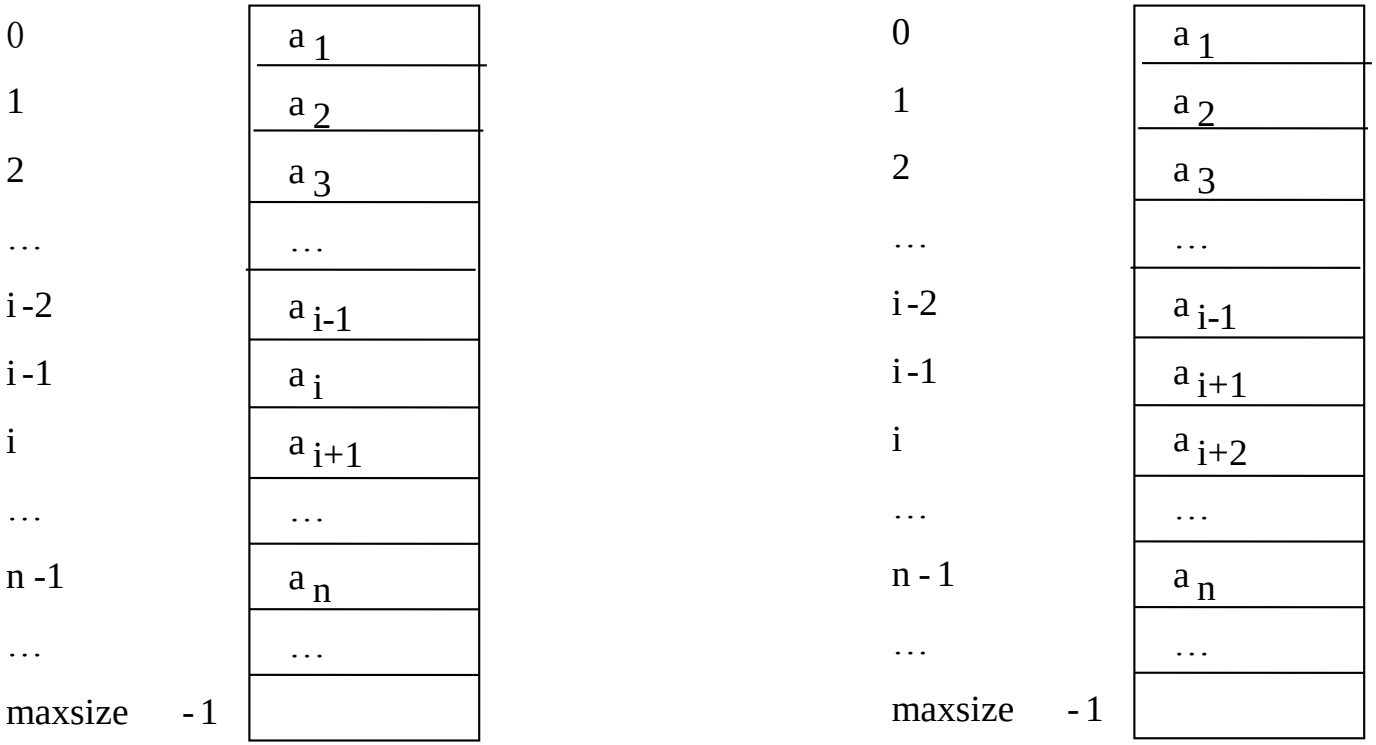


图 2-4 顺序表中删除元素前后状态

算法为：

```
Status ListDelete_Sq(Sqlist &L，int i, ElemType e) {
    //在顺序线性表L中删除第i个元素，并用e返回其值
    // i 的合法值为1<=i<=ListLength_Sq(L);
    if( i<1 || i>L.length)   return ERROR;
    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;     //表尾元素的位置
                                  或可为：  q=&(L.elem[L.length -1]);
    for(++ p; p <=q ; ++ p;)     *(p-1) = *p;
                                 //被删除元素之后的元素左移
    - - L.length;                 //表长减1
    return OK;
}//ListDelete_Sq
```

插入算法花费的时间，主要在于循环中元素的后移（其它语句花费的时间可以省去），即从插入位置到最后位置的所有元素都要后移一位，使空出的位置插入元素值x。但是，插入的位置是不固定的，当插入位置i= 1 时，全部元素都得移动，需n次移动，当i=n+ 1 时，不需移动元素，故在i位置插入时移动次数为n-i+1，假设在每个位置插入的概率相等为$\frac{1}{n+1}$，则平均移动元素的次数为

$$\sum_{i=1}^{n+1}[\frac{1}{n+1}(n-i+1)] = \frac{n}{2}$$

故时间复杂度为O(n)。

同理，可推出删除运算的平均移动次数为 $\frac{n-1}{2}$ ，故时间复杂度为O(n)。

算法2.7

- 例2-2 已知线性表LA和线性表LB中的数据元素按值非递减有序排列，现要求将LA和LB归并为一个新的线性表LC，且LC中的元素仍按值非递减有序排列。

```
void  MergeList_Sq (SqList La，SqList Lb，SqList &Lc){
      //已知顺序线性表La和Lb中的数据元素按值非递减排列。
      //归并La和Lb得到新的顺序线性表Lc, Lc的数据元素也按值非递减排列
      pa = La.elem; pb = Lb.elem;
      Lc.listsize = Lc.length = La.length + Lb.length;
      pc = Lc.elem = (ElemTpye * )malloc(Lc.listsize *  sizeof(ElemType));
      if(!Lc.elem) exit(OVERFLOW);
      pa_last = La.elem + La.length -1;
      pb_last = Lb.elem + Lb.length -1;
      while( pa <= pa_Last && pb <= pb_Last ){        //归并
            if( *pa <= *pb ) *pc++ = *pa++;
            else *pc++ = *pb++;
        }
       while ( pa <= pa_last ) *pc++ = *pa++;  //插入La的剩余元素
       while ( pb <= pb_last ) *pc++ = *pb++; //插入La的剩余元素
)//MergeList_Sq
```

2.2.3 优缺点分析

优点：

a.随机存取，存储位置可用一个简单直观的公式来表示

b.存贮密度高

缺点：

a.删除、插入不方便，需要移动大量元素

b.存贮空间不灵活，且为连续空间

本节作业：

2.11,

2.21

2.3 线性表的链式存贮结构

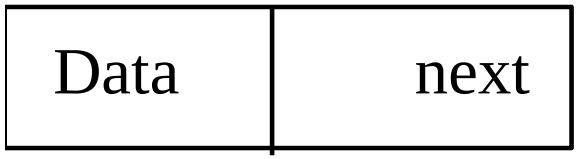
线性表的链式存贮结构，也称为链表。其存贮方式是：在内存中利用存贮单元(可以不连续)来存放元素值及它在内存的地址，各个元素的存放顺序及位置都可以以任意顺序进行，原来相邻的元素存放到计算机内存后不一定相邻，从一个元素找下一个元素必须通过地址(指针)才能实现。故不能像顺序表一样可随机访问，而只能按顺序访问。常用的链表有单链表、循环链表和双向循环链表、多重链表等。

优点：顺序存储结构的缺点

缺点：顺序存储结构的优点

2.3.1 单链表结构（Linklist）

在定义的链表中，若只含有一个指针域来存放下一个元素地址，称这样的链表为单链表或线性链表。线性链表中的结点结构可描述为:



其中data 域用来存放结点本身信息，类型由具体问题而定，本书中，我们也将其设定为elemtype类型，表示某一种具体的已知类型， next域用来存放下一个元素地址。

	头指针	地址	data 域	next 域
head	150	110	a_2	180
		120		
		130	a_4	170
		140	a_6	NULL
		150	a_1	110
		160		
		170	a_5	140
		180	a_3	130

图 2-5 单链表示意图

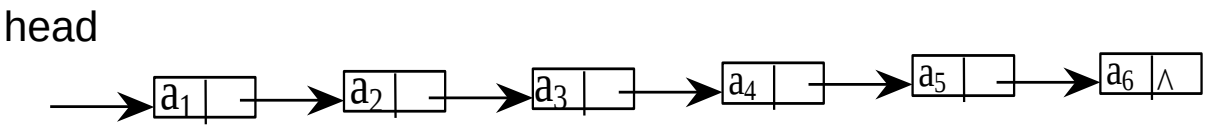
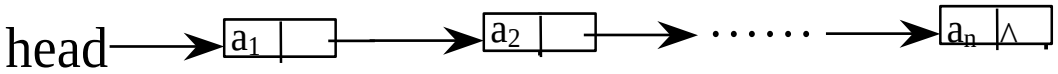


图 2-6 单链表的逻辑表示

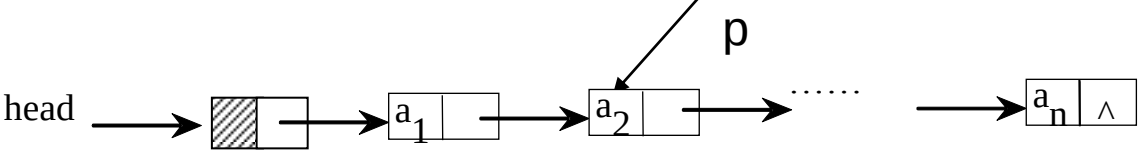
单链表可用类C语言描述为:

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

listnode  *p;
linklist  head;
```



(a) 不带头结点的单链表



(b) 带头结点的单链表

P->data=a_2

图 2-7 不带头结点和带头结点的单链表

注意：

1. 单链表中每个元素的存储位置都包含在其直接前驱结点的信息之中，若p->next=a_i则P->next ->data=a_{i+1} ,因此在单链表中取第i个数据元素必须从头指针出发寻找。（缺点）
2. p为动态变量，它是通过标准函数生成的，即

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

函数malloc分配了一个类型为LinkList的结点变量的空间，并将其首地址放入指针变量p中。一旦p所指的结点变量不再需要了，又可通过标准函数

free(p)

释放所指的结点变量空间。

2.三个名词

首元结点：链表中存储线性表中第一个数据元素的结点。

头指针：指示单链表中第一个结点的存储位置的指针

头结点：为了操作方便,通常在链表的首元结点之前附设的一个结点.

因此，若链表中附设头结点，则不管线性表是否为空，头指针均不为空，否则表示空表的链表的头指针为空。

特点： 数据域为空

作用： 为了对链表进行操作时,可以对空表、非空表的情况以及对首元结点进行统一处理

2．3．2 单链表运算

1． 头插法建立单链表(从左边插入结点)

该方法从一个空表开始，重复读入数据，生成新结点，将读入数据存放到新结点的数据域中，然后将新结点插入到当前链表的表头上，直到读入结束标志为止。

例如：（25,45,18,76,29）之链表的建立过程，因为是在链表的头部插入，读入数据的顺序和线性表中的逻辑顺序是相反的。

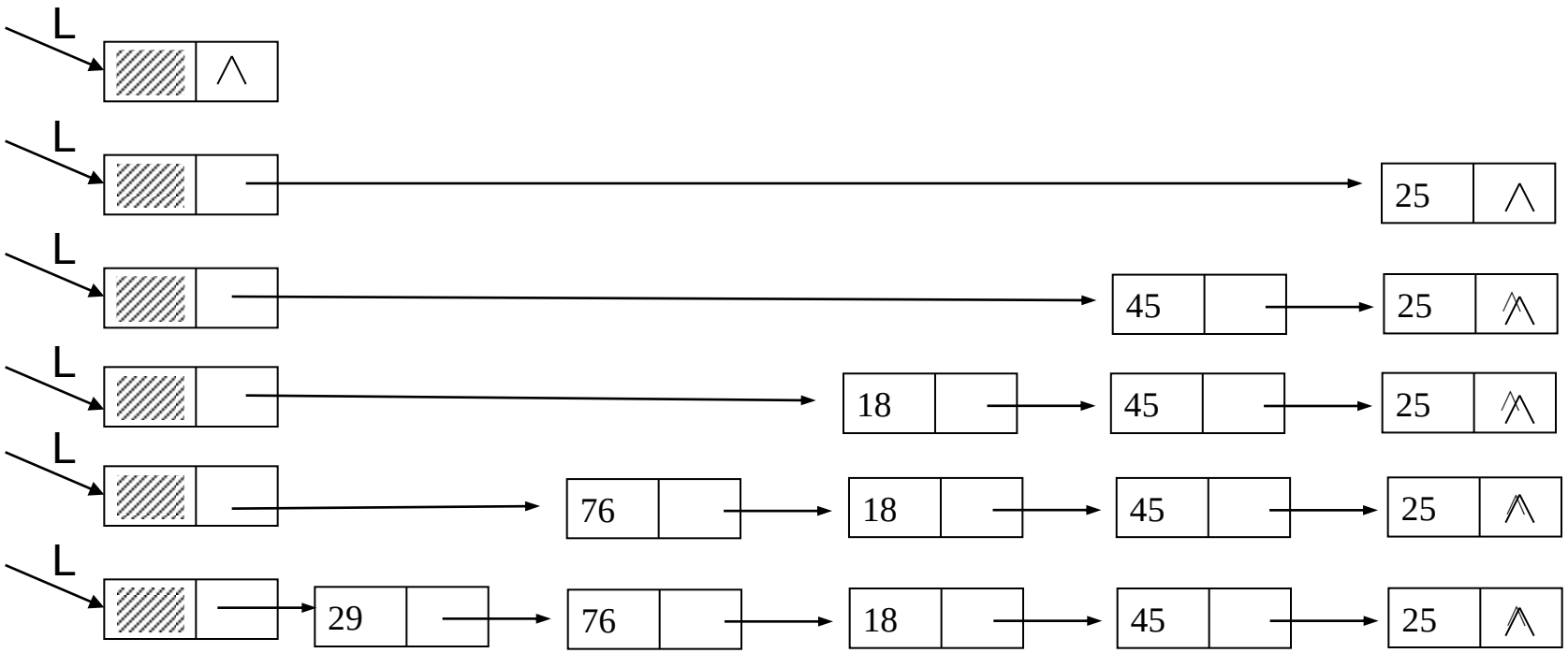


图2.10 在头部插入建立单链表

算法如下：

```
void  CreateList_L(LinkList &L, int n ){
//逆位序输入N个元素的值，建立带表头结点的单链表 L
      L= ( LinkList) malloc(sizeof(LNode));
      L->next = NULL；//空表
      for(i=n; i>0; --i){
          p = ( LinkList) malloc(sizeof(LNode));
          scanf (&p->data);
          p->next = L->next;
          L->next = p;
     }
}//CreateList_L
```

例如：（29,76,18,45,25）之链表的建立过程，如果按照一般方法，按照顺序一个一个插入到表的最后一个位置。

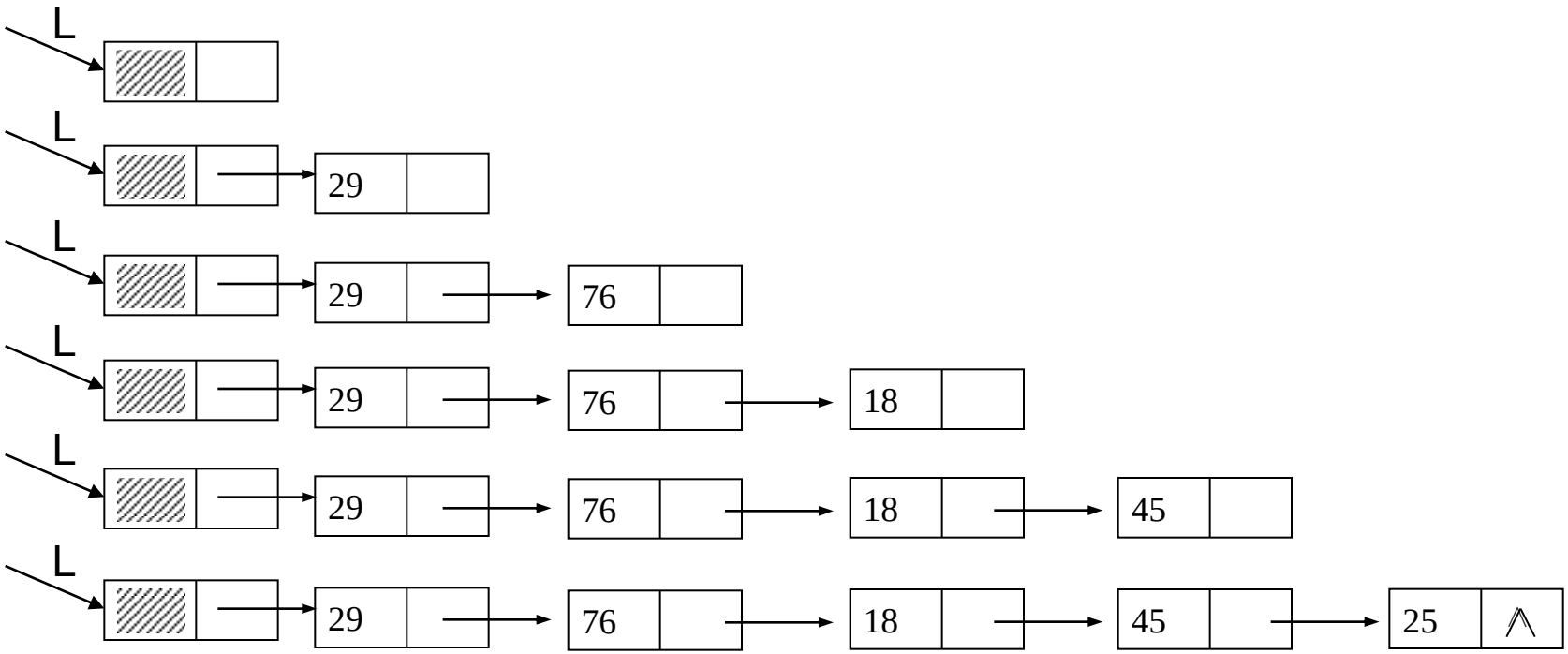


图2.11 一般方法建立单链表

算法如下：

```
void  CreateList_L(LinkList &L, int n ){
//逆位序输入N个元素的值，建立带表头结点的单链表 L
      L= ( LinkList) malloc(sizeof(LNode));
      s = L;
      for(i=0; i<= n-1; i++){
          p = ( LinkList) malloc(sizeof(LNode));
          scanf (&p->data);
          s->next = p ;
          s = p;
     }
     s->next=NULL；//空表
}//CreateList_L
```

2. 单链表上的查找运算

(1) 按值查找

在单链表L中，查找值为x的结点，若找到，返回它的地址，否则返回NULL。

(2) 按序号查找

在单链表L中查找第i个位置上的元素，若找到，则返回它的地址，否则返回NULL。

```
Status void Locate(LinkList &L, ElemType x){
    //在单链表L中，查找值为x的结点，若找到，返回它的地址，
   //否则返回NULL。
    p=L->next;
    while((p!=NULL)&&(p->data!=x))
        p=p->next;
    return p;
}
```

算法的时间复杂度都为O(n)。

```
Status GetElem_L(LinkList  &L, int I,ElemType &e){
    //在单链表L中查找第i个位置上的元素，若找到，则返回它的地址
    p=L->next; j=1;
    while((j<i)&&p){
          j++;
          p=p->next;
    }
     if(!p || j>i)   return Error;
     e=p->data;
     return p;
}
```

算法的时间复杂度都为O(n)。

4. 单链表上的插入运算

若将x插入a和b 之间，插入结点的指针变化如图2 -8所示 。

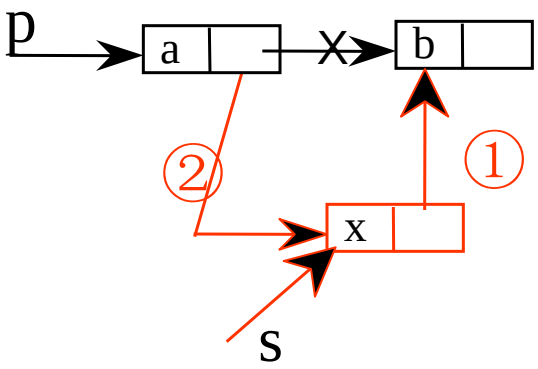


图 2-8 插入结点时的指针修改

```
Status ListInsert_L(linkList  &L, ElemType x ,int i){
   //在带头结点的单链表L中第i个位置之前插入元素
   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或者大于表长
   s=(LinkList)malloc(sizeof(LNode));//生成新结点
   s->data=e;
   s->next=p->next;
   p->next=s;
   return OK;
}//ListInsert_L
```

5． 单链表上的删除运算

若将x删除，删除结点的指针变化如图2-9所示。

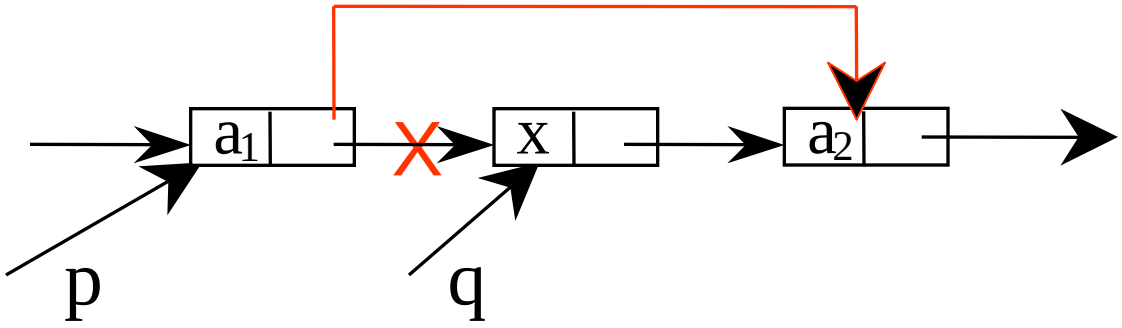


图 2-9 删除结点的指针修改

```
Status ListDelete_L(LinkList  &L, int i, ElemType &e){
    //在带头结点的单链表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){
        p=p->next; ++j;
    }
    if(!(p->next) || j>i-1)  return error;
    q=p->next;   p->next = q->next;
    e=q->data;  free(q);
    return OK;
}//ListDelete_L
```

算法2.12

- 例2-2 已知线性表LA和线性表LB中的数据元素按值非递减有序排列，现要求将LA和LB归并为一个新的线性表LC，且LC中的元素仍按值非递减有序排列。

```
void  MergeList_L (LinkList La，LinkList Lb，LinkList &Lc){
      //已知单链线性表La和Lb中的数据元素按值非递减排列。
      //归并La和Lb得到新的单链线性表Lc, Lc的数据元素也按值非递减排列
      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){          //归并
          if( pa->data <=  pb->data ) {
              pc->next = pa; pc = pa; pa = pa -> next;
           }
          else { 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_Sq
```

本节作业：

自做： 2.6、2.7、2.8

作业本： 2.9、2.13、2.23

2.3.3 静态链表（SLinkList）

静态链表中的结点结构可描述为:

为了 便于在不设“指针”类型的高级程序语言中是使用链表结构，用一个一维数组来描述一个链表，当中的一个分量表示一个结点，用游标（指示器cur）代替指针来指示结点在数组中的相对位置。数组中的第0个分量看成头结点，其游标指向第一个结点。

```
//线性表的静态单链表存储结构
#define   MAXSIZE    1000   //链表的最大长度
typedef  struct{
      ElemType  data;
      int cur;
}//component, SLinkList[MAXSIZE]
```

若指示第一个结点：s[0].cur=i,则第一个结点的值为s[i].data,第二个结点的位置为s[i].cur.

则右图静态链表的数据的逻辑顺序为：（A，B，C，D，E，F）

用游标代替指针：

q=p ->next;（动态链表）

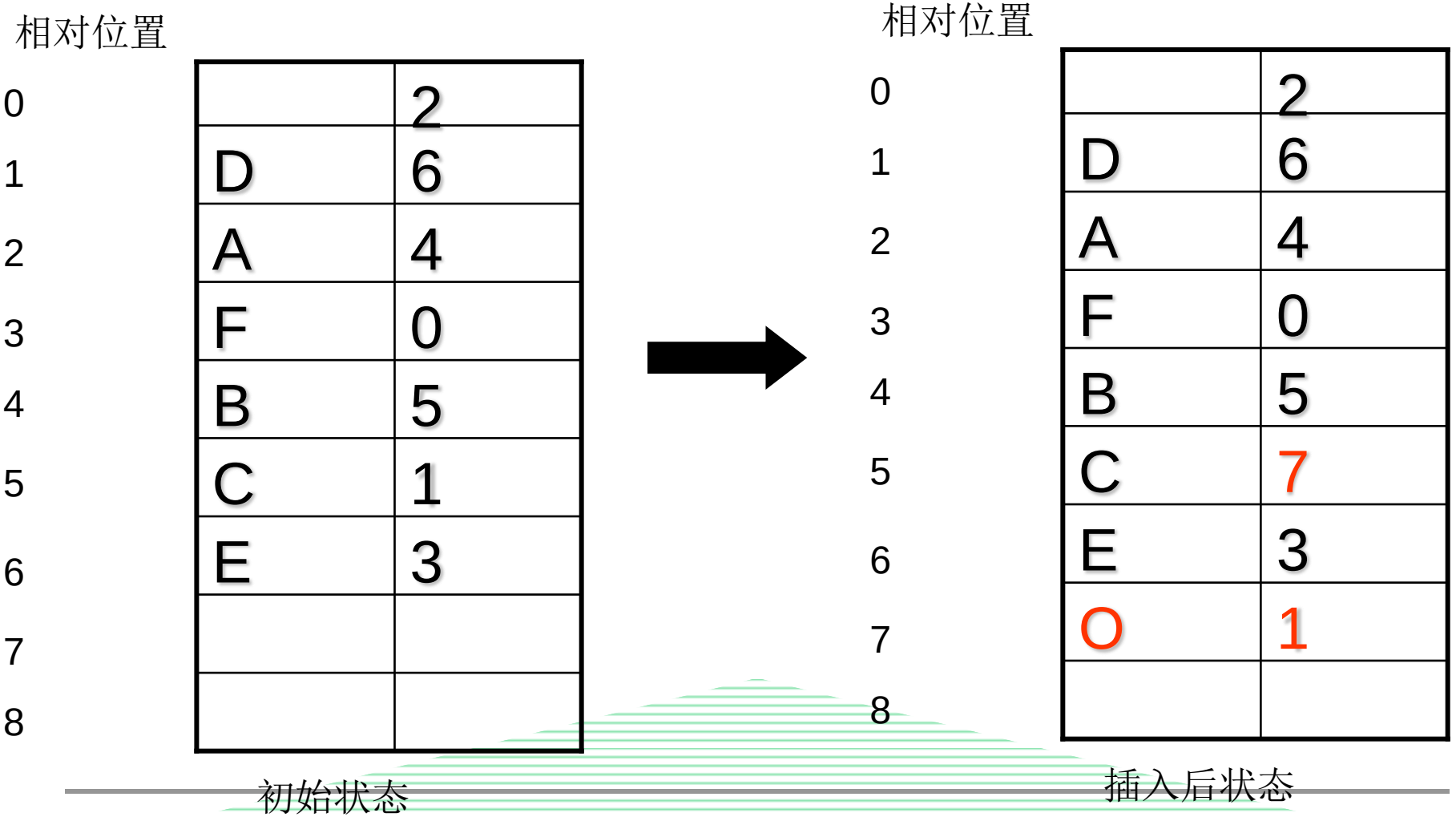
k=s[i].cur;（静态链表）

相对位置	data	cur
0		2
1	D	6
2	A	4
3	F	0
4	B	5
5	C	1
6	E	3
7		
8		

静态链表示例

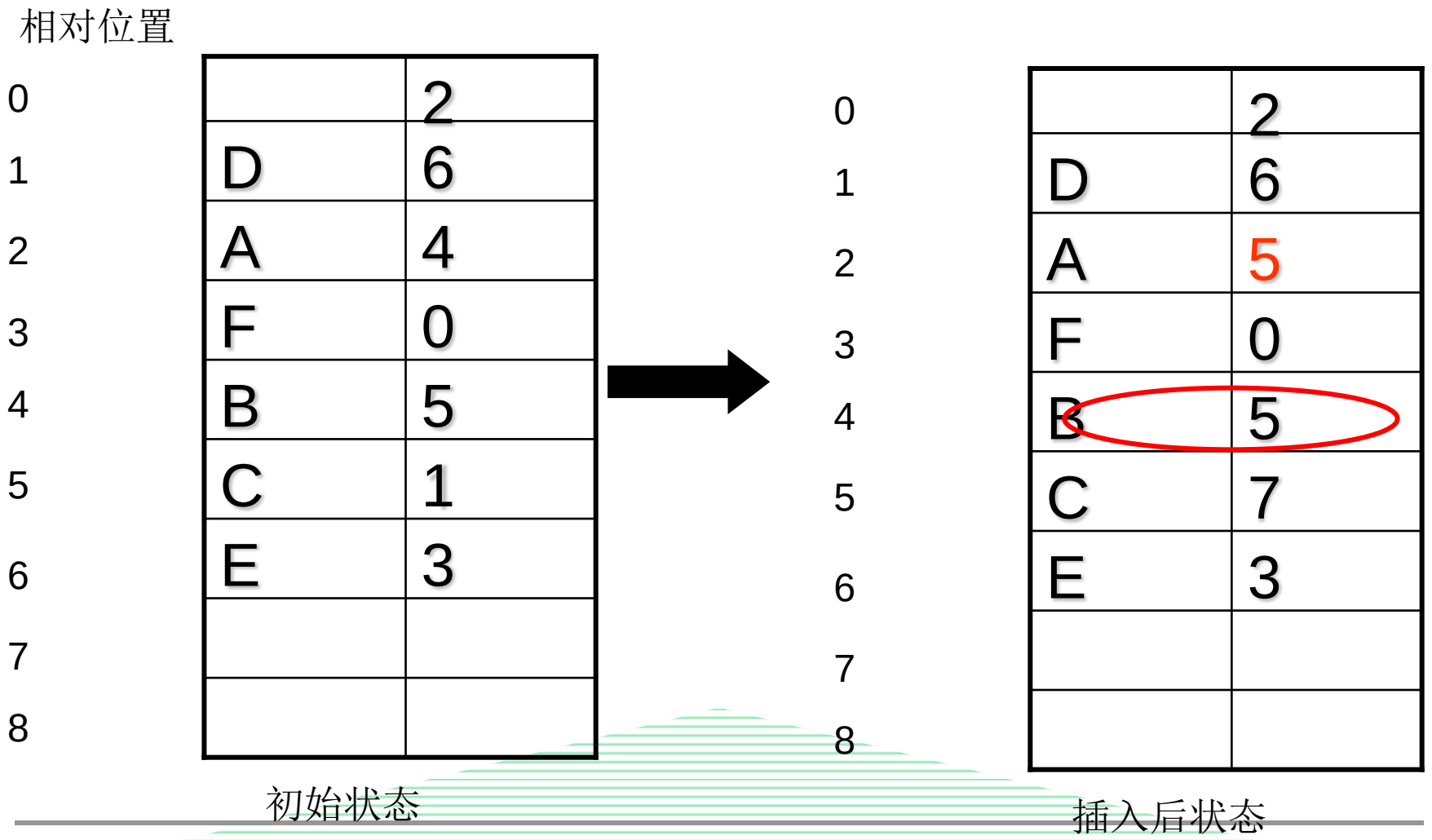
静态链表的插入

例：在（A，B，C，D，E，F）的数据C和D之间插入数据O



静态链表的插入

例：删除（A，B，C，D，E，F）中的数据B

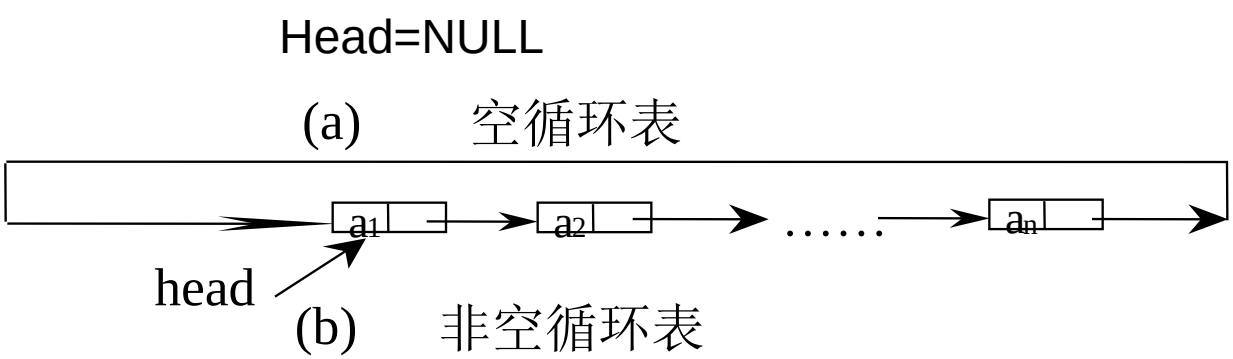


结论：

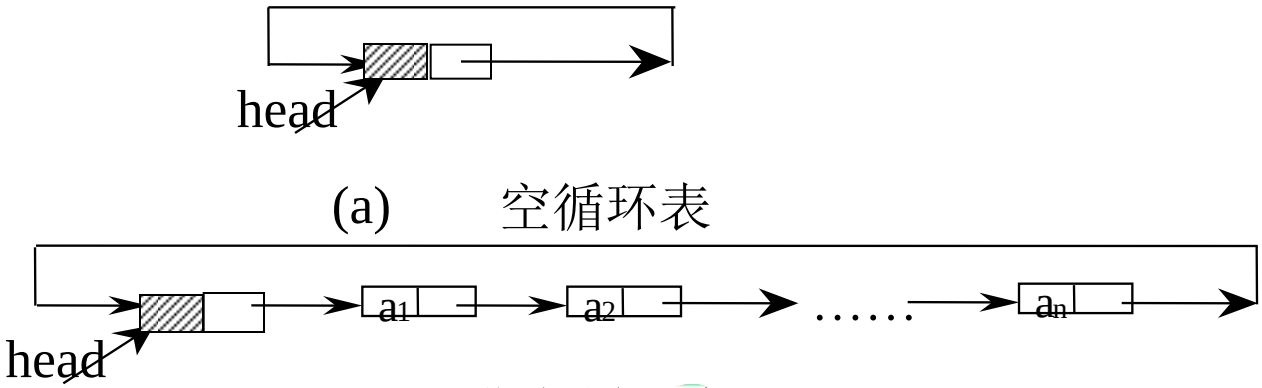
静态链表中实现线性表的操作和动态链表相似，以整型游标代替动态指针，插入和删除不需移动元素，仅需修改游标值，仍具有链式存储结构的主要优点。

2.3.3 循环链表结构

单链表上的访问是一种顺序访问，从其中某一个结点出发，可以找到它的直接后继，但无法找到它的直接前驱。因此，我们可以考虑建立这样的链表，具有单链表的特征，但又不需要增加额外的存贮空间，仅对表的链接方式稍作改变，使得对表的处理更加方便灵活。从单链表可知，最后一个结点的指针域为NULL表示单链表已经结束。如果将单链表最后一个结点的指针域改为存放链表中头结点(或第一个结点)的地址，就使得整个链表构成一个环，又没有增加额外的存贮空间，称这们的链表为单循环链表，在不引起混淆时称为循环表(后面还要提到双向循环表)



无头结点的单循环链表示意图



带头结点的单循环链表示意图

循环链表的操作

1.删除

无头结点：分两种情况

a.删除第一个结点

b.删除除第一个结点外的其余结点

有头结点：与单链表操作相似

2.插入　有无头结点操作基本一致

注意：

- 由于循环链表中没有NULL指针，故涉及遍历操作时，其终止条件就不再像非循环链表那样判断p或p—>next是否为空，而是判断它们是否等于某一指定指针，如头指针等。

  从头结点出发遍历所有结点，循环结束的条件：

  单链表：p->next ==NULL;

  循环链表：p->next ==head;

- 在很多实际问题中，表的操作常常是在表的首尾位置上进行，此时头指针表示的单循环链表就显得不够方便.如果改用尾指针rear来表示单循环链表，则查找开始结点a_1和终端结点a_n都很方便，它们的存储位置分别是(rear–>next) —>next和rear，显然，查找时间都是O(1)。因此，实际中多采用尾指针表示单循环链表。

例、在链表上实现将两个循环线性表(a_1，a_2，a_3，…a_n)和(b_1，b_2，b_3，…b_n)链接成一个线性表的运算。

```
status  connect(CiLinkList taila，CiLinkList tailb){
    //taila和tailb分别是带头结点的循环链表La和Lb的尾结点
     p=taila—>next;
     taila—>next=(tailb—>next)—>next
     free(tailb—>next);
     tailb—>next=p;
     return(tailb);
}//connect
```

2.3.4 双向链表结构

1. 双向链表基本概念

在单链表中，从某个结点出发可以直接找到它的直接后继，时间复杂度为O(1) ，但无法直接找到它的直接前驱；在单循环链表中，从某个结点出发可以直接找到它的直接后继，时间复杂仍为O(1)，直接找到它的直接前驱，时间复杂为O(n)。有时，希望能快速找到一个结点的直接前驱，这时，可以在单链表中的结点中增加一个指针域指向它的直接前驱，这样的链表，就称为双向链表(一个结点中含有两个指针)。如果每条链构成一个循环链表，则会得双向循环链表。

双向链表结点的定义如下：

```
typedef  struct  DuLNode{
        ElemType data;
        struct   DuLNode *prior,*next;
}DuLNode,*DuLinkList;
```

双向链表结点

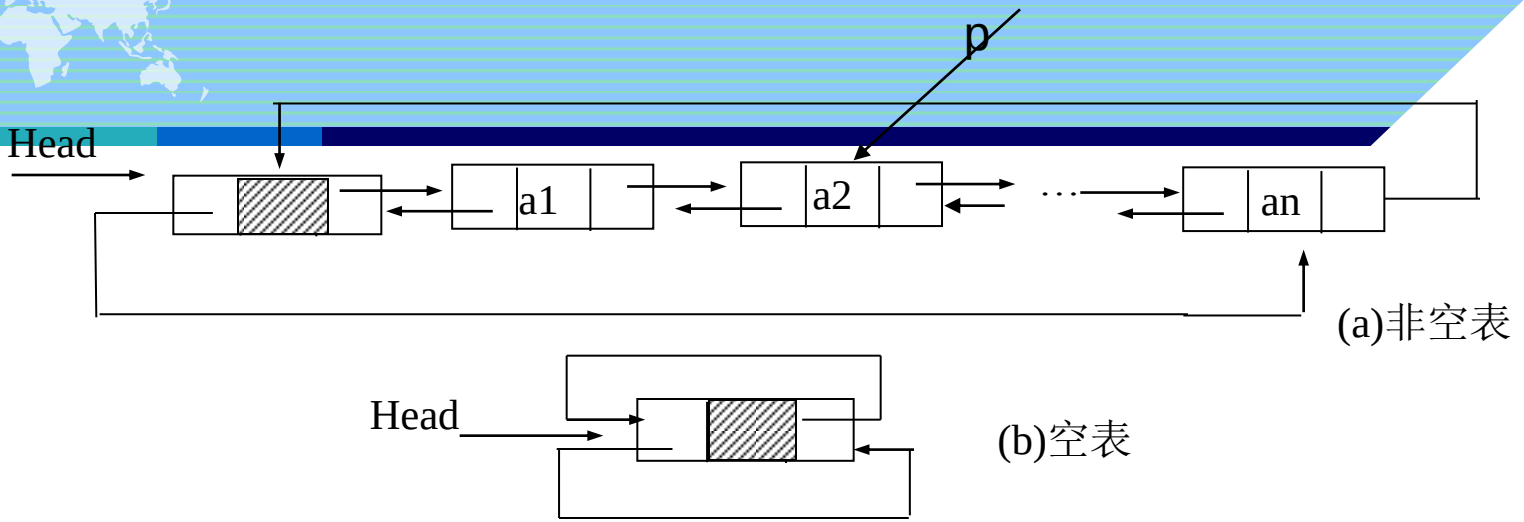


图2.19 带头结点的双循环链表

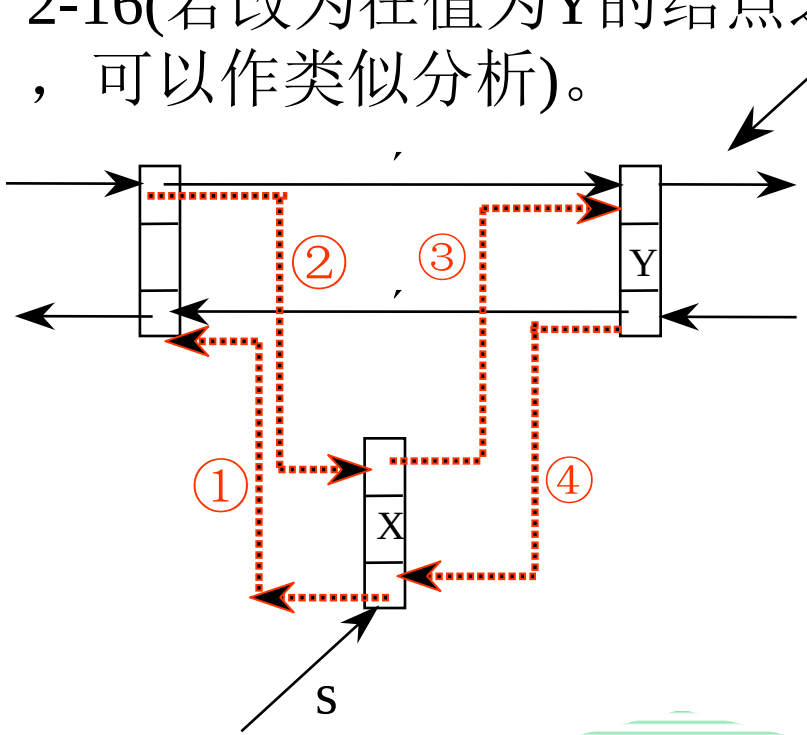
设p指向双向循环链表中的某一结点，即 p中是该结点的指针，则p->prior->next表示的是*p结点之前驱结点的后继结点的指针，即与p相等；类似，p->next->prior表示的是*p结点之后继结点的前驱结点的指针，也与p相等，所以有以下等式：

p->prior->next = p = p->next->prior

2. 双向链表的运算

1）插入

在head为头指针的双向链表中，在值为Y的结点之前插入值为X的结点，插入结点的指针变化见图2-16(若改为在值为Y的结点之后插入值为X的结点，可以作类似分析)。



① s->prior=p->prior;
② p->prior->next=s;
③ s->next=p;
④ p->prior=s;

图2-16 S 插入P之后指针变化示意图

插入算法描述为：

```
Status ListInsert_DuL(DuLinkList  &L, int i, ElemType e){
    //在带头结点的双链循环线性表L中第i个位置之前插入元素e，
    //i的合法值为1<=i<=表长+1
    if( ! (p=GetElemP_DuL(L,i) ) )  //在L中确定第i个元素的位置指针
        return  ERROR;                    //p=NULL，第i个元素不存在
    if( ! (s=( DuLinkList )malloc( sizeof(DuLNode) ) ) )     return ERROR;
    s->data=e;
    s->prior=p->prior;   p->prior->next = s;
    s->next = p; p->prior = s;
    return OK;
}//ListInsert_DuL
```

3．双向链表的删除运算

在以head为头的双向链表中删除值为x的结点，删除算法的指针变化见图2-17。

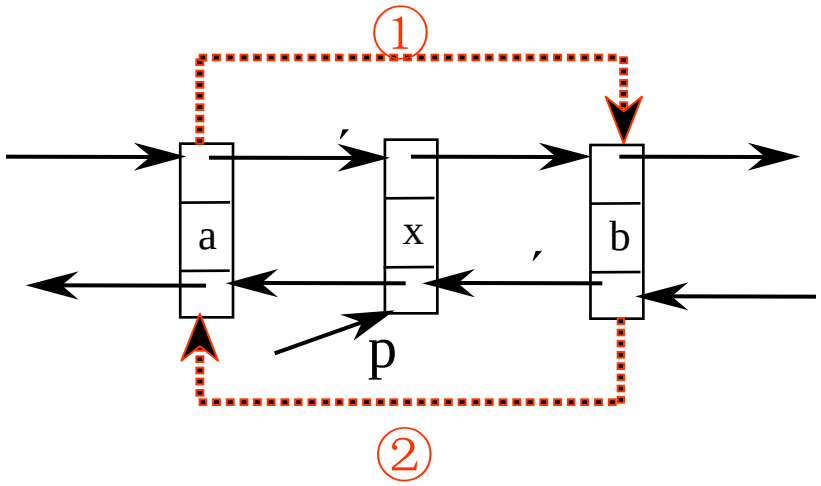


图 2-17　删除P指针所指结点示意图

删除算法描述为:

```
Status ListDelete_DuL(DuLinkList &L, int i, ElemType &e){
    //删除带头结点的双链循环线性表L的第i个元素，i的合法值为
    //1<=i<=表长
    if( ! (p=GetElemP_DuL(L,i) ) )  //在L中确定第i个元素的位置
        return  ERROR;              //p=NULL，第i个元素不存在
    e=p->data;
    p->prior->next=p->next;
    p->next->prior=p->prior;
    free(p);
    return OK;
}// ListDelete_DuL
```

四、链表的应用举例

1。一元多项式的表示及相加

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

$Q_m(x)=q_0+q_1x+q_2x^2+\ldots+q_nx^m$

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

用线性表来表示为：

$P = (p_0, p_1, p_2, \ldots, p_n)$

$Q = (q_0, q_1, q_2, \ldots, q_m)$

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

2. 年级学生成绩管理系统

附加作业1

在一个年级学生成绩管理系统中，希望处理各班信息以及各班每个学生的学习情况信息，其中班级信息包括班号和名称，学生学习情况信息包括学号、姓名、班号等，以及已学课程的课程号以及成绩，并能使管理人员通过界面完成对班级、学生信息的录入以及对数据的查找、浏览。

年级学生成绩管理系统----实现功能

- 提供用户界面
- 登记各班学生基本情况（如学号、姓名、性别、年龄、电话等）
- 插入某班某个学生的基本情况
- 修改各班学生基本情况
- 删除某班某个学生或某班所有学生的基本情况
- 登记各班所有学生各门功课的成绩
- 修改某个学生某门功课的成绩
- 浏览各个班级信息
- 查找、浏览每个学生的基本信息
- 查找、浏览每个学生的全部成绩信息

年级学生成绩管理系统 ----系统提示

- 分四个功能模块实现年级学生成绩管理系统
- 班级管理模块：主要实现的功能为添加、删除某个班级。
- 学生管理模块：主要实现的功能为登记、修改某班某个学生的基本情况、删除某班某个学生的基本情况。
- 成绩管理模块：主要实现的功能为登记、修改某个学生某门课的成绩。
- 查询、浏览模块：主要实现的功能为浏览各个班级信息；查找、浏览每个学生的基本信息；查找、浏览每个学生的全部成绩信息。

实验一 单链表的就地反转

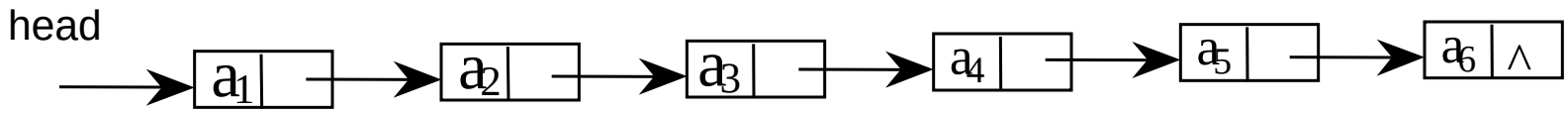
实验内容：单链表的就地反转

实验时间：10月17号

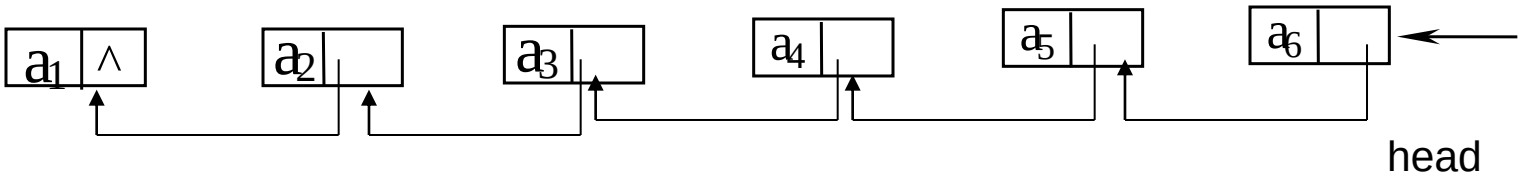
实验地点：10-409、413、414

实验一 单链表的就地反转

原链表：



就地反转后的链表：



```
Status invert_list( link &head)  {
   //对带头指针的单链表实行就地反转
   back=head->next;
   p=back->next;
   back->next=NULL;

   while(p!=NULL)
   {
      q=p->next;
      p->next=back;
      back=p;
      p=q;
   }

    head->next=back;

    return  head;

}// invert_list
```

```
/*单链表的反转*/
link invert_list( link head) {
     struct list *back,*p,*next;

     back=head->next;
     p=back->next;
     back->next=NULL;

     while(p!=NULL) {
         q=p->next;
         p->next=back;
         back=p;
         p=q;
     }
    head->next=back;

    return  head;
}
```

实验一 单链表的就地反转

完整程序应包括4个算法：

- 新建单链表
- 依次输出单链表的结点值
- 就地反转
- 摧毁单链表

实验一 单链表的就地反转

完整程序应包括4个算法：

- 新建单链表
- 依次输出单链表的结点值
- 就地反转
- 摧毁单链表