第二章　线性表

王　　勇
计算机\软件学院　大数据分析与信息安全团队
21#518　　电　话 13604889411
Email：　wangyongcs@hrbeu.edu.cn

哈爾濱工程大學
Harbin Engineering University

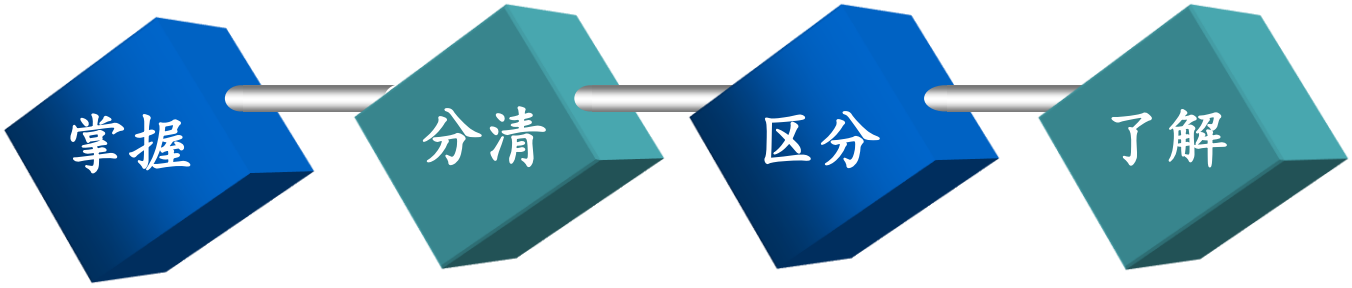
本章内容

1. 线性表的类型定义
2. 线性表的顺序表示和实现
3. 线性表的链式表示和实现
4. 循环链表和双向链表
5. 一元多项式的表示及相加

本章主要内容

知识点	重　点	难　点
线性表 顺序表 链　表 有序表	顺序表 链　表	链　表 算法设计题

基本要求



掌握	分清	区分	了解
扎实的指针操作和内存动态分配的编程技术	分清链表中指针 p 和结点 *p 之间的对应关系	区分链表中的头结点、头指针和首元结点的不同	循环链表、双向链表的特点

了解特性

熟练掌握

综合比较

增强理解

逻辑结构特性，即数据元素间存在着线

能够从时间和空间复杂度的角 比较线性表 储 及其适用场

结合线性表类型的定义增强对抽象数据类型的理解。

学习目标

线 性 表

是$n \geq 0$个数据元素a_1，a_2，…，a_{i-1}，a_i，a_{i+1}，…，a_n的有限序列。

可将线性表记为(a_1，a_2，…，a_{i-1}，a_i，a_{i+1}，…，a_n)

线性表中的数据元素是属于同一数据对象，相邻数据元素之间存在着序偶关系。（同构元素）

一个数据元素可以是由若干个数据项组成的记录，含有大量记录的线性表称为文件。

线性表的类型定义

例 如 学校的学生健康情况登记表。

姓　名	学　号	性别	年龄	班级	健康状况
王小林	10061101	男	18	计10611	健康
陈　红	10061102	女	20	计10611	一般
刘建平	10061103	男	21	计10611	健康
张立立	10061104	男	17	计10611	神经衰弱
⋮	⋮	⋮	⋮	⋮	⋮

线性表的类型定义

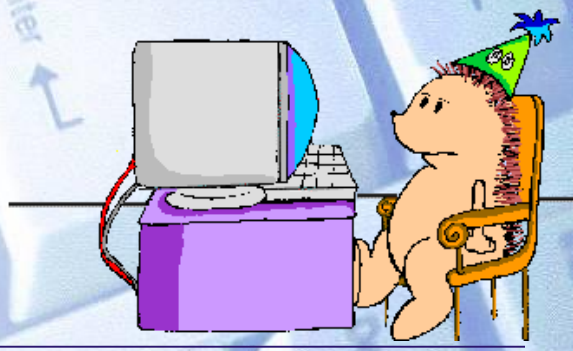
例 如

英文字母（A，B，C，……，Z）

表中数据元素是一个字母。

星期（星期日，星期一，星期二，……，星期六）

表中数据元素是星期中一天的名称。



线性结构的特点

- 存在唯一的一个被称做“第一个”的数据元素a_1
- 存在唯一的一个被称做“最后一个”的数据元素a_n
- 除第一个之外，每个元素都只有一个直接前驱

  a_{i-1}是a_i的直接前驱
- 除最后一个之外，每个元素都只有一个直接后继

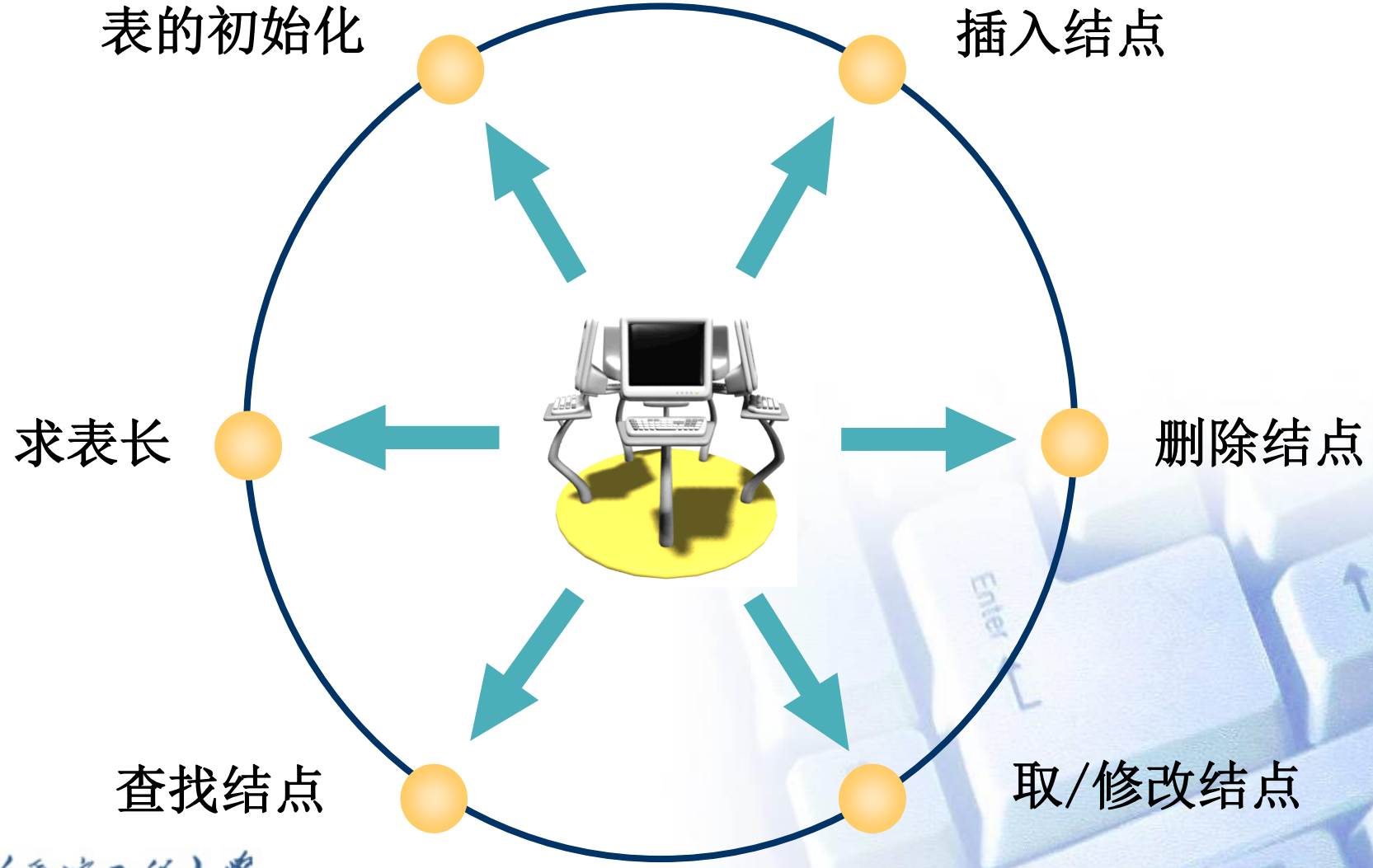
  a_{i+1}是a_i的直接后继
- i是数据元素a_i在线性表中的位序

线性表的长度

线性表中数据元素的个数n，当n=0时，为空表。

线性表的类型定义

线性表的基本运算



ADT List{

数据对象：$D=\{a_i|a_i \in ElemSet, i=1, 2, \ldots, n, n>=0\}$

数据关系：$R1=\{<a_{i-1}, a_i>|a_{i-1}, a_i \in D, i=1, 2, \ldots, n\}$

基本操作：

InitList(&L)

//创建一个空的线性表L

DestroyList(&L)

//撤消L

ClearList(&L)

//将L重置为空表

ListEmpty(L)

//判L是否为空? 空为T

ListLength(L)

//返回表长度(元素个数)

GetElemList(L,i,&e)

//用e返回L中第i个数据元素的值

已经实现了上述定义的线性表类型，就可以在应用问题的求解中利用这些定义的各种操作

例2-1　两个线性表LA、LB，将存在于线性表LB中而不在LA中的数据元素加入到线性表LA中。即 LA=LA∪LB

算法的思想

逐一取出LB中的元素，判断是否在LA中，若不在，则插入LA 。

（1）先求出LA、LB长度

（2）若LB未取空，取LB中一元素e，若LB取空则转（4）

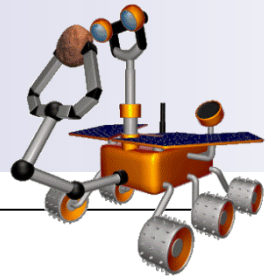
（3）如e不在LA中，则插入到LA中，转（2）

（4）结束

```
void unin(List &La，List Lb)
{   La_len=(ListLength(La));
    Lb_len=(ListLength(Lb));
    for (i=1; i<=Lb_len; i++)
        {  GetElem(Lb, i, &e);          //取LB中第i个元素
           if (!LocateElem(La, e, equal)) //La_len次
               ListInsert(&La, ++La_len, e);
        }       //La 中不存在和e 相同的元素，则插入之
 } //union
```

算法的时间复杂度

O(ListLength(LA)×ListLength(LB))



例2-2　线性表LA和LB是非递减有序的，将两表合并成新的线性表LC，且LC也是非递减的。

算法的思想

将LA、LB两表中的元素逐一按序加入到一个新表LC中。
分别设三个变量，i和j、k，指向各表的相应元素

（1）LA、LB均不空转（2），否则转（4）

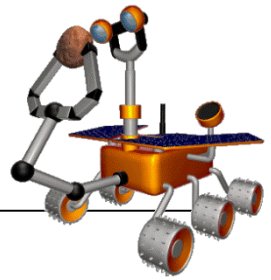
（2）在LA中取一元素=>a，LB中取一元素=>b

（3）若a<=b, a==>LC, 否则b==>LC, 转（1）

（4）若LA不空，则LA剩余部分放到LC尾

（5）若LB不空，则LA剩余部分放到LC尾

```
void MergeList(List La, List Lb, List &Lc)
{
    InitList(Lc);     //建一空表LC
    i=j=1; k=0;      //i, j, k分别指向La, Lb, Lc初始位置
    La_len=(ListLength(La));
    Lb_len=(ListLength(Lb));
    while (i<=La_len) && (j<=Lb_len) //La和Lb均非空
        {  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)   //La还有元素
    {  GetElem(La, i++, ai);
       ListInsert(Lc, ++k, ai);
    }
 while (j<=Lb_len) //Lb还有元素
    {   GetElem(Lb, j++, bj);
       ListInsert(Lc, ++k, bj);
    }
}//MergeList
```

算法的时间复杂度

O(ListLength(LA)+ListLength(LB))

本章内容

1. 线性表的类型定义
2. 线性表的顺序表示和实现
3. 线性表的链式表示与实现
4. 循环链表和双向链表
5. 一元多项式的表示及相加

线性表的顺序表示和实现

1. 线性表的顺序表示
2. 顺序表的特点
3. 线性表的顺序存储结构
4. 顺序线性表的操作
5. 顺序表的优缺点

顺序表　用一组地址连续的存储单元存储一个线性表的数据元素

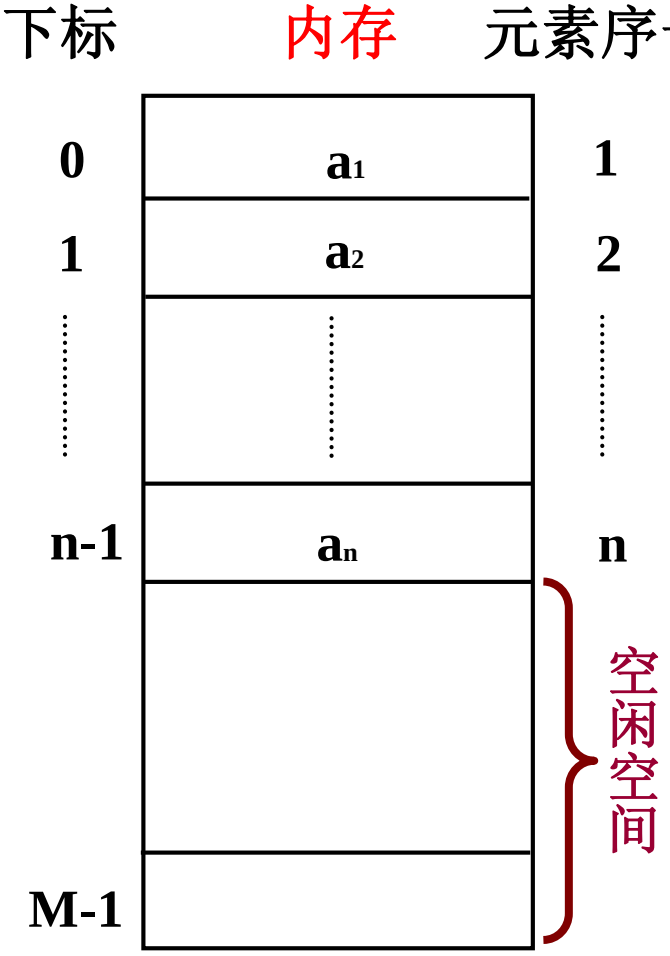
表　示　一维数组

存储地址

- 元素存储位置：$LOC(a_{i+1})=LOC(a_i)+L$
- 一般存储位置：$LOC(a_i)=LOC(a_1)+(i-1)*L$

其中：L—一个元素占用的存储单元个数

$LOC(a_i)$—线性表第i个元素的地址




```
typedef int DATATYPE;
#define M 1000
DATATYPE   data[M];
```

```
例  typedef  struct  card
        {   int num;
            char name[20];
            char author[10];
            char  publisher[30];
            float  price;
        } DATATYPE;
    DATATYPE  library[M];
```

动态申请和释放内存

```
DATATYPE *pData = (DATATYPE *)malloc(M*sizeof(DATATYPE));
free(pData);
```

逻辑上相邻—物理地址相邻

随机存取

表 示　用一维数组定义一个线性表

```
#define LIST_INIT_SIZE 100
#define LISTINCREAMENT 10
typedef struct
  {    ElemType *elem;  //指向线性表起始地址的指针
       int  length;          //线性表实际存放数据长度
       int  listsize;        //线性表申请长度
 }SqList;
```

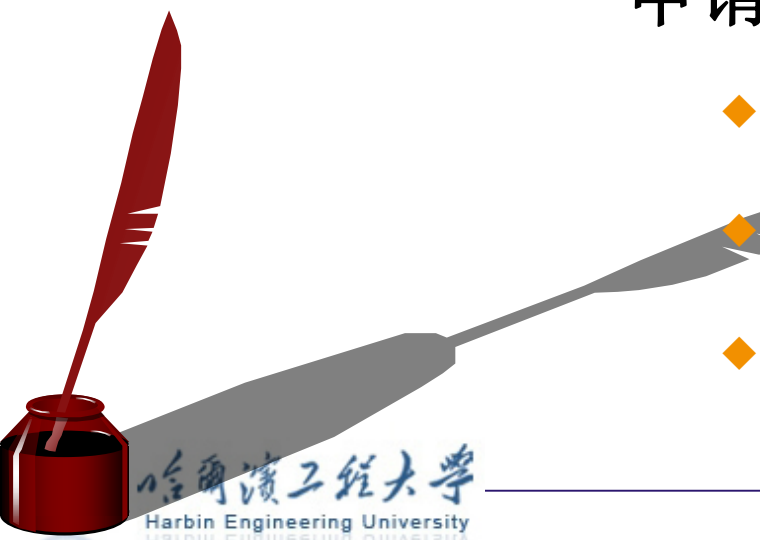
顺序表容易实现访问操作，可随机存取元素。但插入和删除操作主要是移动元素。

初始化　构造一个空顺序表

算法思想

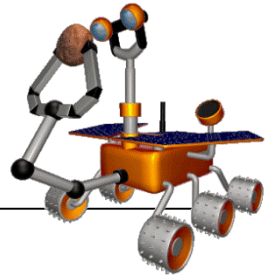
申请存储空间，设置表

- 起始地址 **elem**
- 表长 **length**
- 可用空间 **listsize**



顺序表初始化算法2.3

```
Status InitList_Sq(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;
}//InitList_Sq
```



插　入

线性表的插入是指在第i（1≤i ≤ n+1）个元素之前插入一个新的数据元素x，使长度为n的线性表

(a1, a2, …, ai-1, ai, ai+1, …, an)

变成长度为n+1的线性表

(a1, a2, …, ai-1, x, ai, ai+1, …, an)

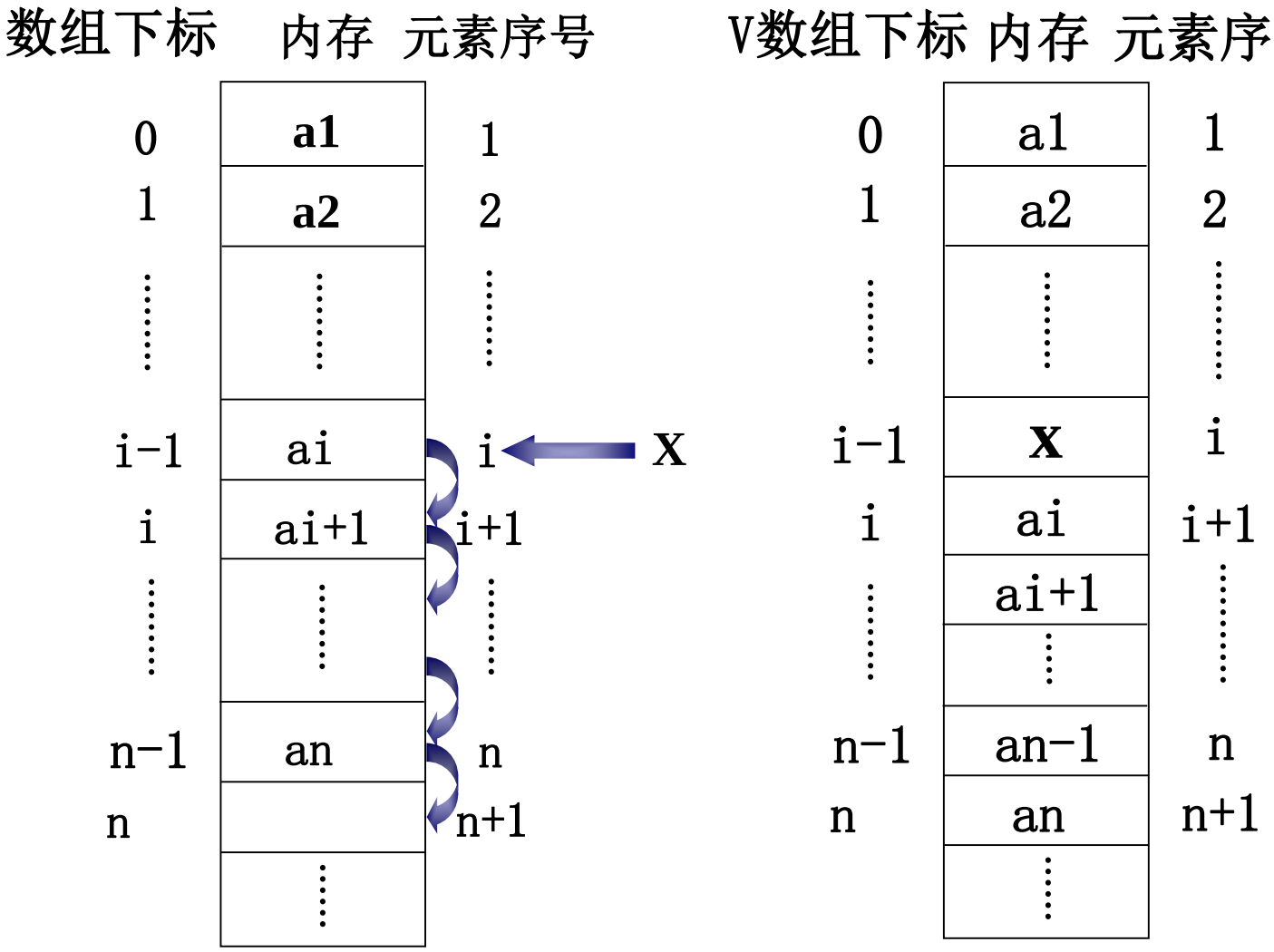
需将第i至第n共（n-i+1）个元素后移

Click

算法思想

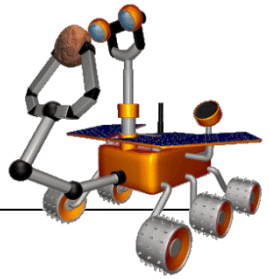
- 判i值的合法性，1≤i≤表长+1
- 判表的空间满否？若满则增加动态分配单元
- 从表元素n到i，依次后移一个位置
- 将e插入第i个位置，表长度增1

next


数组下标
内存
元素序号
0
1
i-1
i
n-1
n
a1
a2
ai
ai+1
an
1
2
i
i+1
n
n+1
X
V数组下标
内存
元素序号
0
1
i-1
i
n-1
n
a1
a2
X
ai
ai+1
an-1
an
1
2
i
i+1
n
n+1

插入算法2.4

```
Status ListInsert_Sq(SqList &L, int i, ElemType e )
{  if (i<1||i>L.length+1) return ERROR;      //i不合法
   if (L.length>=L.listsize)                  //可用空间已满增加分配
      {  newbase=(ElemType*)realloc(L.elem,
                 (L.listsize+LISTINCREMENT)sizeof(ElemType));
         if (!newbase) exit(OVERFLOW);
         L.elem=newbase;          //新基址
         L.listsize+= LISTINCREMENT;
      }
   q=&(L.elem[i-1]);               //取插入位置q
   for (p=&(L.elem[L.length-1]); p>=q; --p)
         {  *(p+1)=*p;   }         //将n~i位置的元素后移
   *q=e;                           //插入e
   ++L.length;                     //表长度+1
   return OK;
}//ListInsert_Sq
```



讨　论　插入元素的时间主要花费在移动元素上，而移动元素的个数主要取决于插入的位置。

设pi是在第i个元素之前插入一个元素的概率，则在长度为n的线性表中插入一个元素时需移动元素的平均次数为

$$Eis = \sum_{i=1}^{n+1} P_i (n - i + 1)$$

Click

若认为 $P_i = \frac{1}{n+1}$　等概率

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

$\therefore T(n) = O(n)$

结　论　平均移动表的一半元素，当n很大时，效率很低

删　除

线性表的删除是指将第（i+1）至第n个元素逐一向前移动一个位置，将长度为n

(a1, a2, …, ai-1, ai, ai+1, …, an)

变成长度为n-1的线性表

(a1, a2, …, ai-1, ai+1, …, an)

需将第i至第n共（n-i）个元素前移

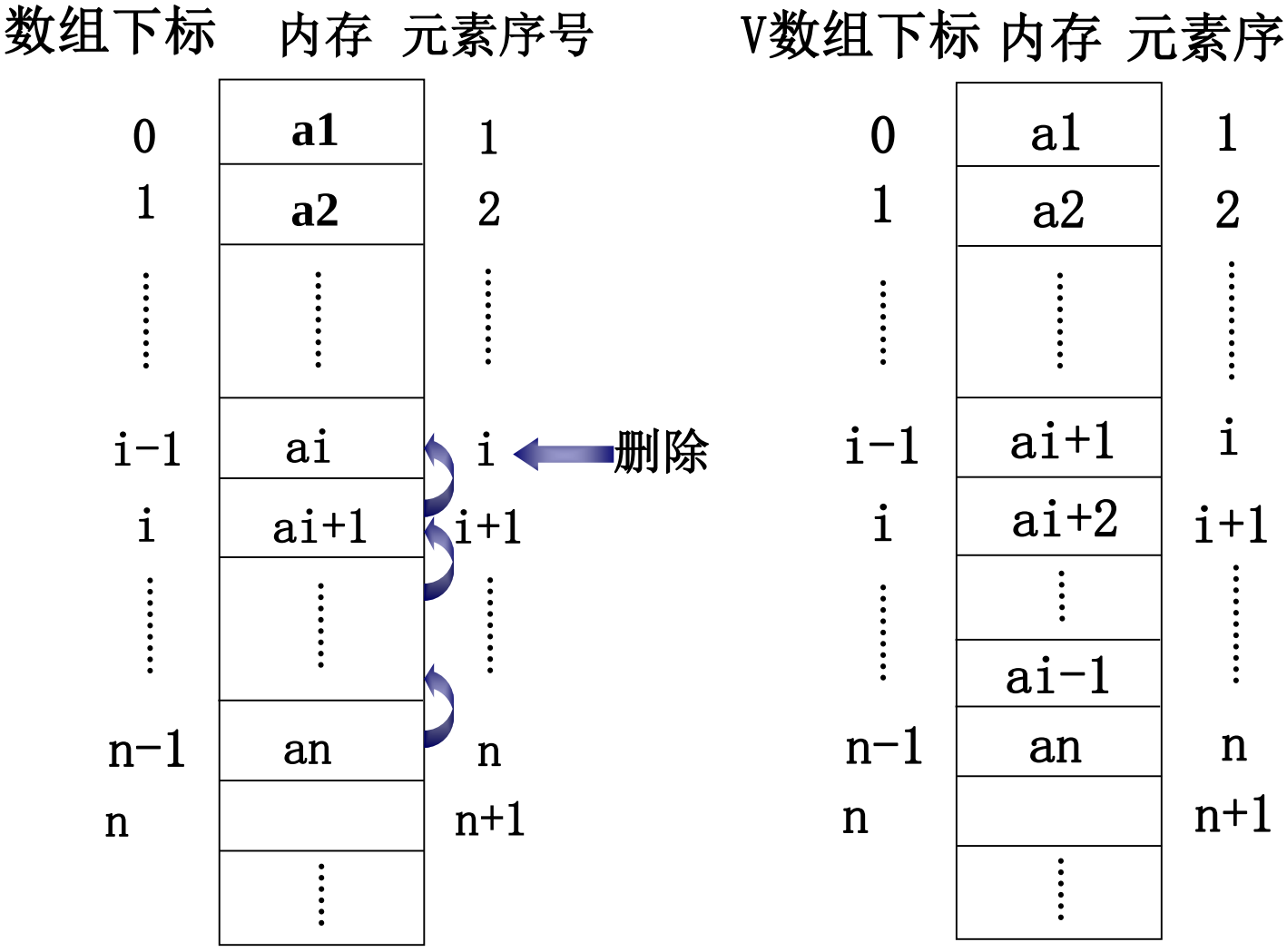
Click

算法思想

在表L中删除第i个元素，放入e

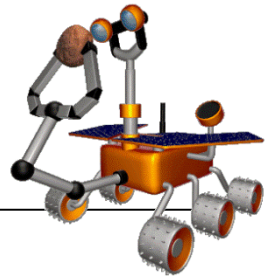
- 判i值的合法性，1≤i≤表长n
- 取第i个元素，放入e
- 从i+1到表长n，依次前移
- 表长度减1

删除


数组下标
内存
元素序号
0
1
i-1
i
n-1
n
a1
a2
ai
ai+1
an
1
2
i
i+1
n
n+1
删除
V数组下标
内存
元素序号
0
1
i-1
i
n-1
n
a1
a2
ai+1
ai+2
ai-1
an
1
2
i
i+1
n
n+1

删除算法2.5

```
Status ListDelete_Sq(SqList &L, int i, ElemType &e )
{  if ( (i<1) || (i>L.length) ) return ERROR;
                                        //i不合法
   p=&(L.elem[i-1]);                    //取被删除元素位置
   e=*p;                                //取插入位置元素的值
   q=L.elem+L.Length-1;   //取表尾元素的位置
   for (++p; p<=q; ++p;)
       {  *(p-1)=*p;   }  //将i+1~n位置的元素前移
   --L.length;                                  //表长度-1
   return OK;
}//ListDelete_Sq
```



讨　论　删除元素的时间主要花费在移动元素上，而移动元素的个数主要取决于删除的位置。

设qi是删除第i个元素的概率，则在长度为n的线性表中删除一个元素所需移动的元素次数的平均次数为

Click

$$E_{de}=\sum_{i=1}^{n}Q_i(n-i)$$

若认为$Q_i=\frac{1}{n}$　等概率

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

$\therefore T(n)=O(n)$

结　论　平均移动表的一半元素，当n很大时，效率很低

顺序表的的优缺点

优点

逻辑相邻即物理相邻

可随机存取任一元素

存储空间使用紧凑

缺点

插入删除移动元素量大

预分配空间利用不充分

容量难扩充

本章内容

1. 线性表的类型定义
2. 线性表的顺序表示和实现
3. 线性表的链式表示和实现
4. 循环链表和双向链表
5. 一元多项式的表示及相加

线性表的链式存储结构

1. 线性链表的表示
2. 线性链表的特点
3. 线性表的链式存储结构
4. 链式线性表的操作
5. 链式表的优缺点

线性链表

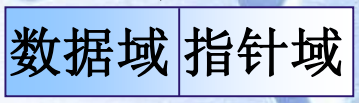
由n个结点，链结成一个链表，称为线性链表或单链表

用指针实现元素（或关系）逻辑上的相邻结点（Node）：数据域、指针域、指针、链、头指针

链表实现

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

结点



例如 线性表

(ZHAO,QIAN,SUN,LI,ZHOU,WU,ZHENG,WANG)

存储地址	数据域	指针域
1	LI	43
7	QIAN	13
13	SUN	1
19	WANG	NULL
25	WU	37
31	ZHAO	7
37	ZHENG	19
43	ZHOU	25

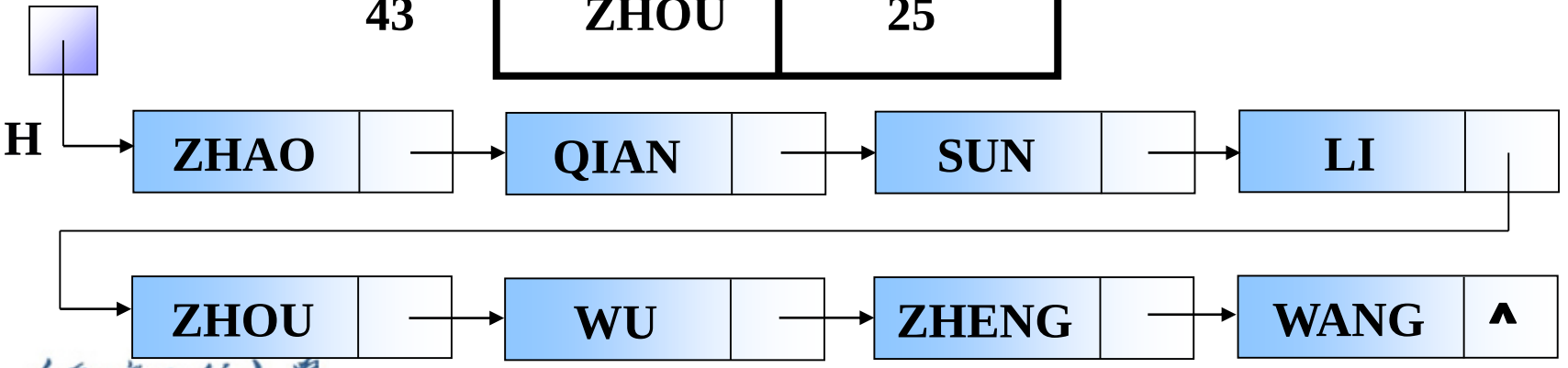
头指针

H

31

数据之间的逻辑关系由结点中的指针指示

最后结点



顺序表的特点

逻辑上相邻
—物理不相
邻

不能随机
存取

注 意

LNode *L,*p;

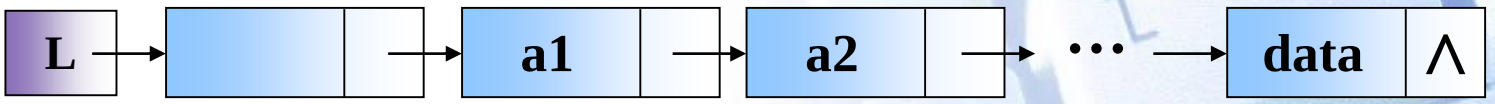
结点(*p)

- (*p)表示p所指向的结点，p是指向结点(*p)的指针
- p->data表示p指向结点的数据域
- p->next表示p指向结点的指针域
- 生成一个LNode型新结点：

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

- 系统回收p结点：free(p);

带头结点非空表

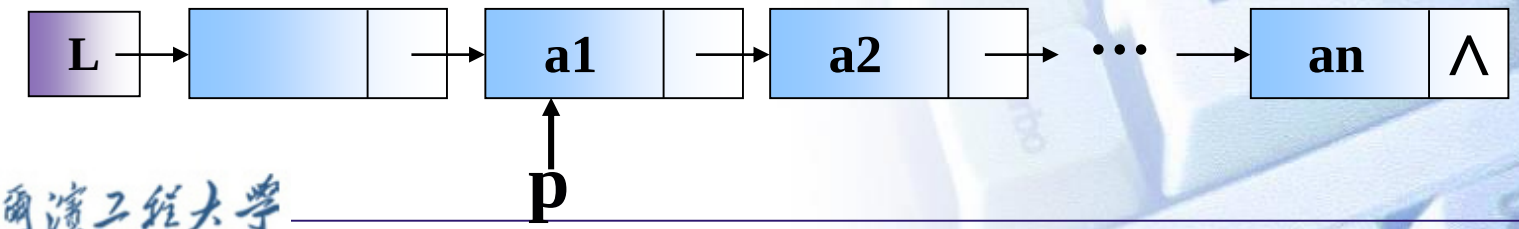


带头结点空表

取结点值 取带头结点的单链表的第i个元素的值，L为头指针

算法思想

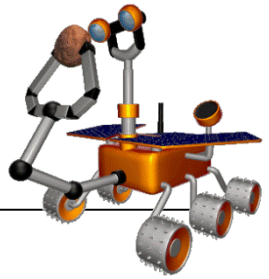
- 如果链表不为空，做如下操作
- 指针p指向第1个结点
- 设一计数器j初值为1
- j<i且p不为空，重复做
  - p指针后移，计数器j++
  - 直到j=i或p为空



单链表取结点算法2.8

```
Status GetElem_L(LinkList L, int i, ElemType  &e)
{  //L为带头结点的单链表的头指针。
   //当第i个元素存在时，其值赋给e并返回OK，
     否则返回ERROR
    p=L->next;                    //p指向第1个结点，j做计数器
    if  (p) //直接判断初始化时p是否为空
       { j=1;  //再找i-1个
         while(p && j<i)    //直到p指向第i元素或p为空
              {  p=p->next; ++j;}
         if (p)  { e=p->data;  return OK;} //p为真即j=i

         else  return ERROR
       }
    else
       return ERROR
}//GetElem_L
```

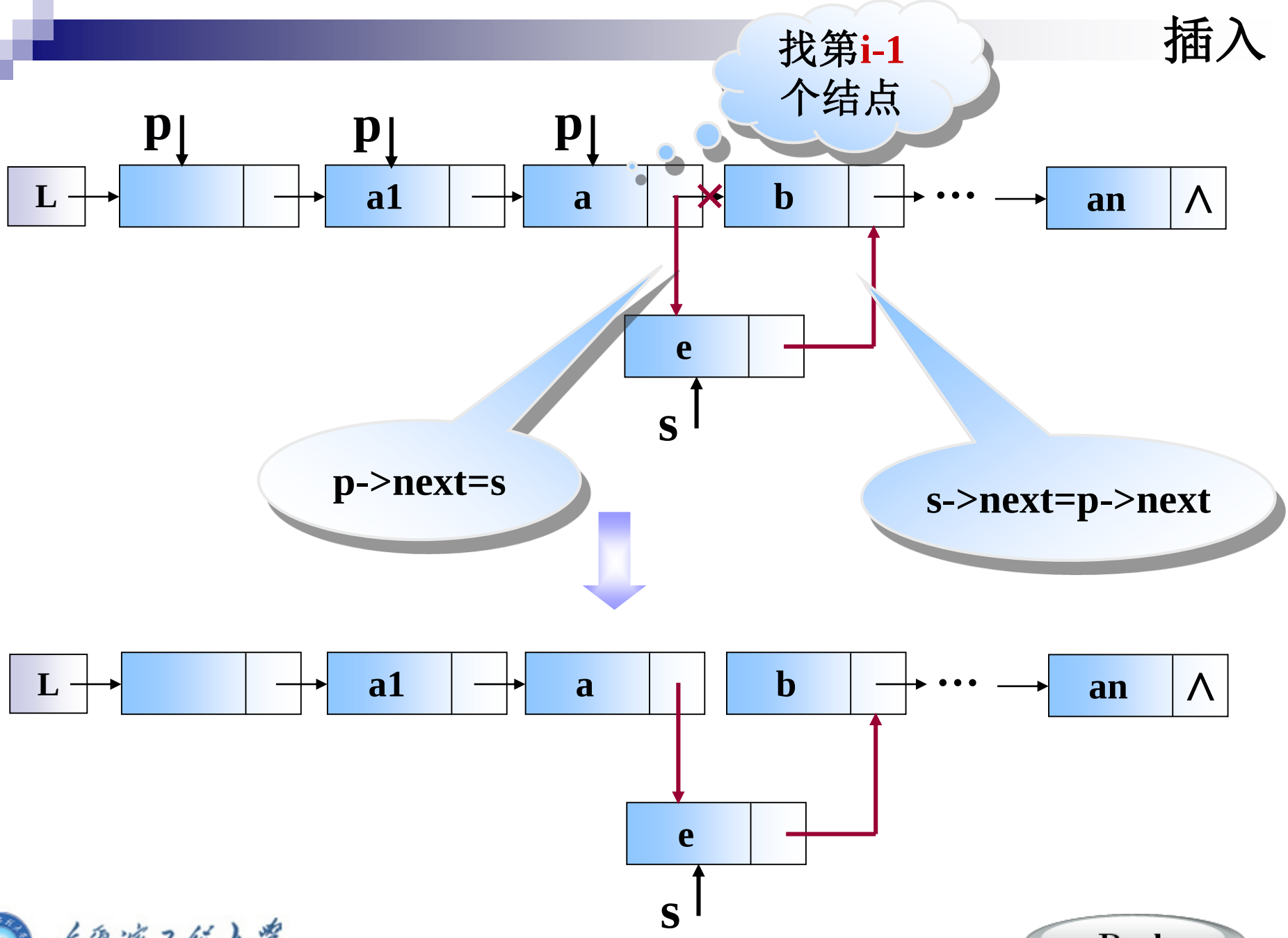


插入结点 在带头结点的单链表L中第i个位置之前插入元素e（即在元素a和b 之间插入元素e）

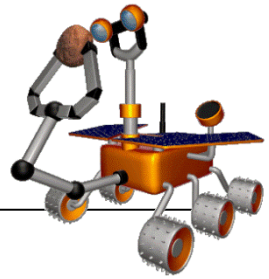
算法思想

Click

- 设p为指向头结点 指针，s为指向结点e的指针
- 先找到第i-1个位置（即p指向元素a）
- 生成新结点e，s指针指向新结点
- 插入，修改s、p的指针
- 让a的next指针指向e，且e的next指针指向b。实现三个元素a、e和b的逻辑关系


找第i-1个结点
p
p
p
L
a1
a
b
an
e
s
p->next=s
s->next=p->next
L
a1
a
b
an
e
s

```
Status ListInsert_L(ListLink &L,  int i, ElemType e)
{ //在带头结点的单链表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; }  //找第i-1结点
    if (!p || j>i-1) return ERROR;   //i小于1或大于表长
    s=(LinkList*)malloc(sizeof(LNode));
    s->data=e;
    s->next=p->next;
    p->next=s;
    return OK;
}//ListInsert_L
```



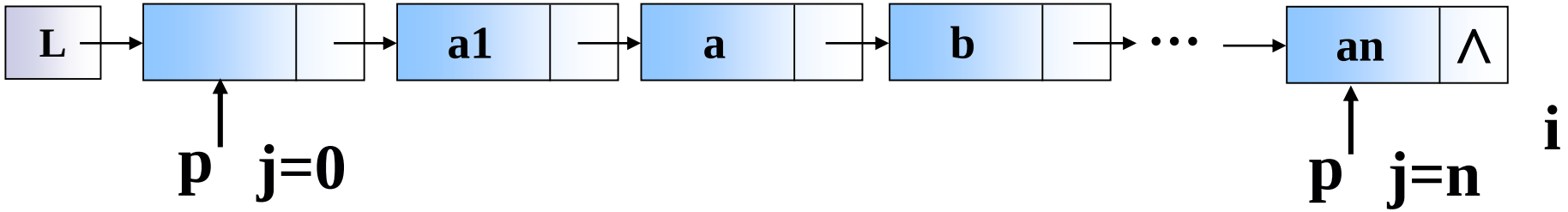
删除结点 在带头结点的单链表L中删除第i个元素，并由e返回

算法思想

Click

- 设p为指向头结点指针
- 先找到第i-1个位置（即p指向元素a）
- s指向第i个结点
- 删除，修改p的指针
- 让p的next指针指向e，且e的next指针指向b。实现三个元素a、e和b的逻辑关系

next

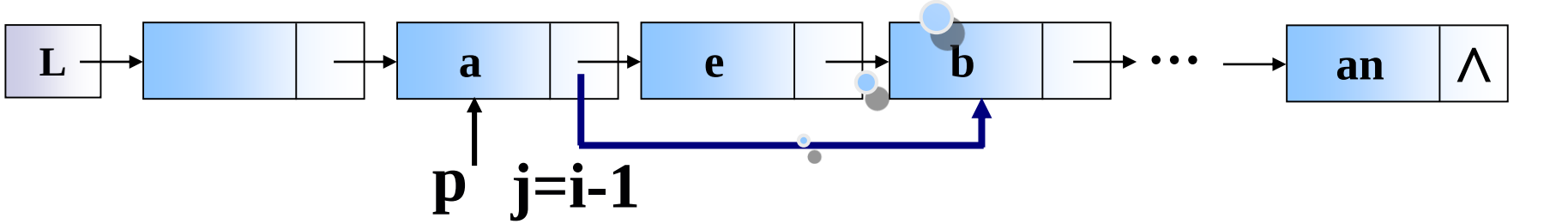


非法删除情况：

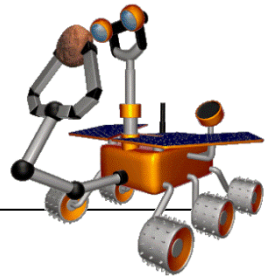
i>表长n：p->next=null

i<1： 即i=0,-1,-2….., j>i-1

p->next=
p->next-next



```
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
   p=L;  j=0;
   while (p->next && j<i-1) { p=p->next; ++j; }
         //寻找第i个结点，并令p指向其前驱第i-1结点
   if (!p->next || j>i-1) return ERROR;//i>表长||i<1
   q=p->next; p->next=q->next;
   e=q->data;
   free(q);
   return OK;
}//ListDelete_L
```



动态创建链表

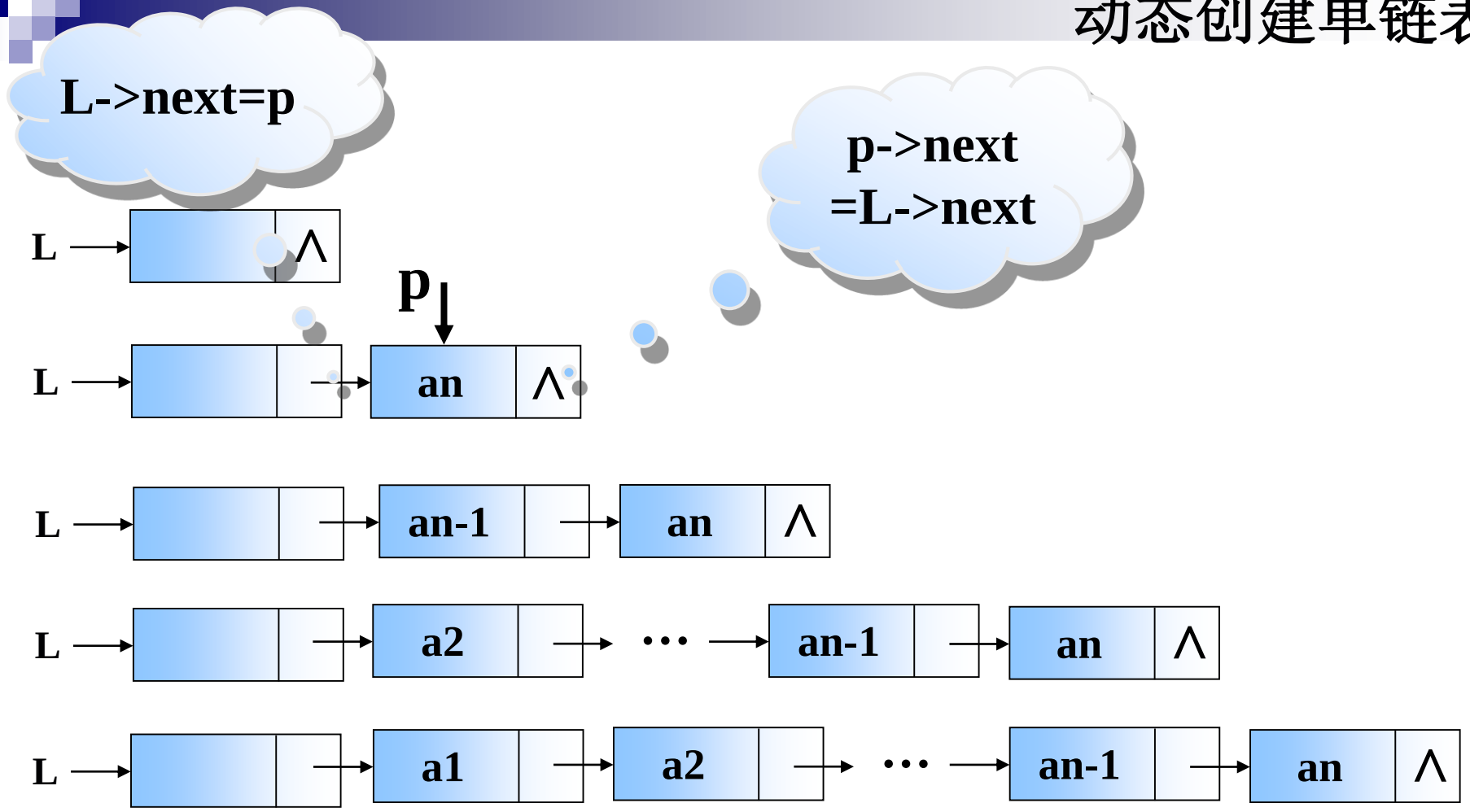
逆位序输入n个元素值，动态的建立带头结点的单链表L，并由L返回头指针

算法思想

Click

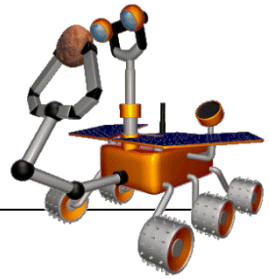
- 申请头结点，设L为指向头结点指针
- 头结点的next为Null
- 重复做下面步骤，直到最后一个结点
- 申请结点，输入结点ai值
- 在头结点之后插入结点

next


L->next=p
p->next
=L->next
L
∧
p
L
an
∧
L
an-1
an
∧
L
a2
…
an-1
an
∧
L
a1
a2
…
an-1
an
∧

动态建立单链表算法2.11

```
Void CreateList_L(LinkList &L, int n)
  {     //逆位序输入n个元素值，
        // 建立带头结点的单链表L
  L=（LinkList*)malloc(sizeof(LNode));
  L->next=NULL;
  for (i=n; i>0; --i)
     {
       p=(LinkList*)malloc(sizeof(LNode));
       scanf(&p->data);
       p->next=L->next;  L->next=p;
     }
   }//CreatList_L
```



合并有序链表 将两个有序链表La、Lb合并为一个有序链表Lc， La、Lb、 Lc为头结点指针

算法思想

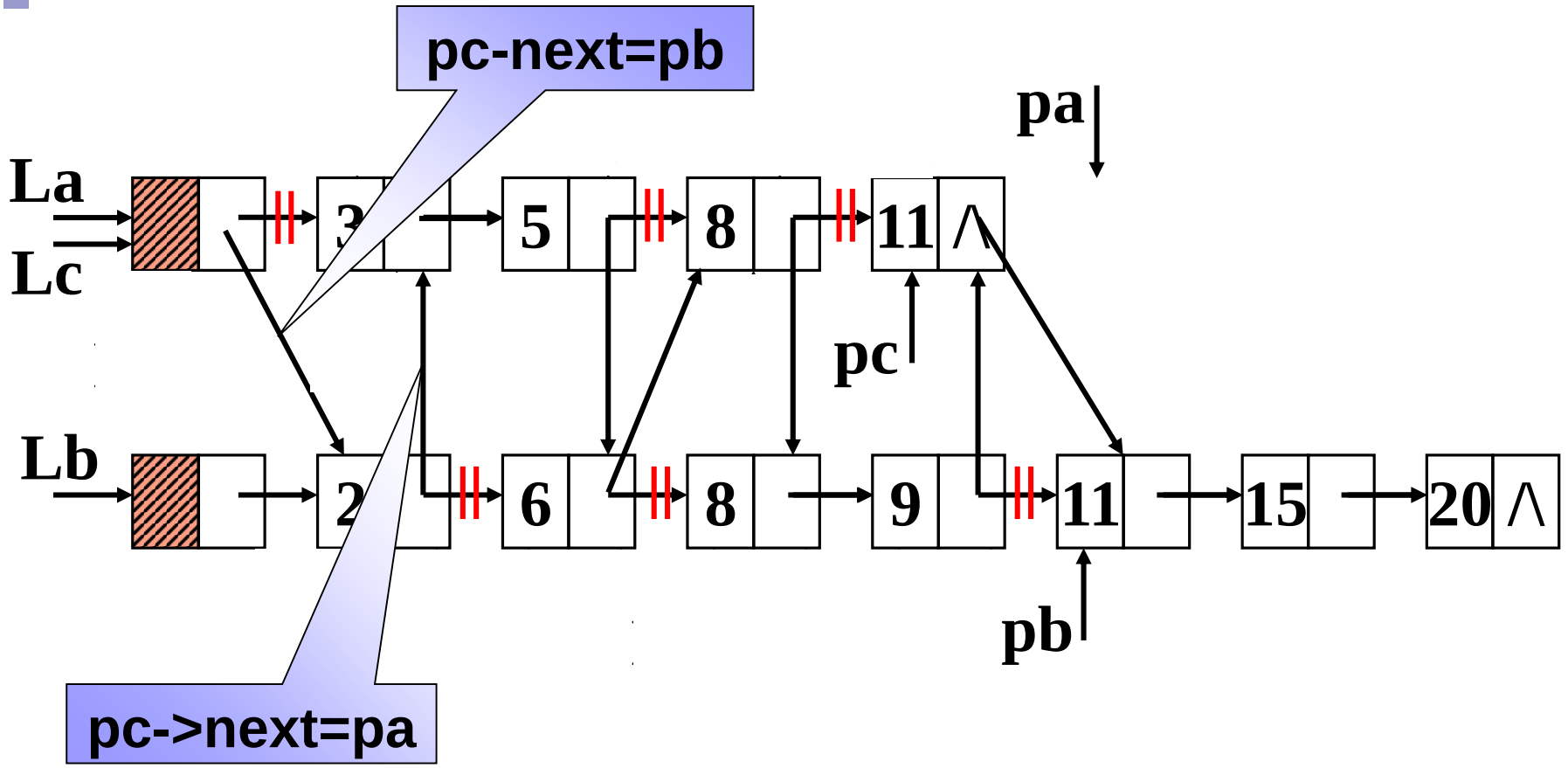
Click

在归并两个链表时，不需另建新表的结点空间，只需将原来两链表中结点之间关系解除，重新建立关系。

- 设立三个指针pa、pb和pc分别用来指向两个有序链表和合并表的当前元素
- 比较两个表的当前元素的大小，将小的元素链接到合并表Lc中,让合并表的当前指针指向该元素，然后，修改指针。

next

合并有序链表



每次链接pa或pb的一个结点后，便做如下操作：

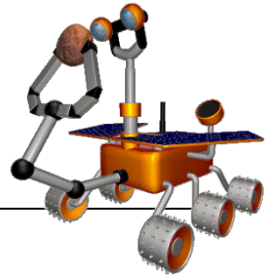
pc=pa; pa=pa->next; pa和pc分别后移一个位置

或pc=pb; pb=pb->next; pb和pc分别后移一个位置

合并有序链表算法2.12

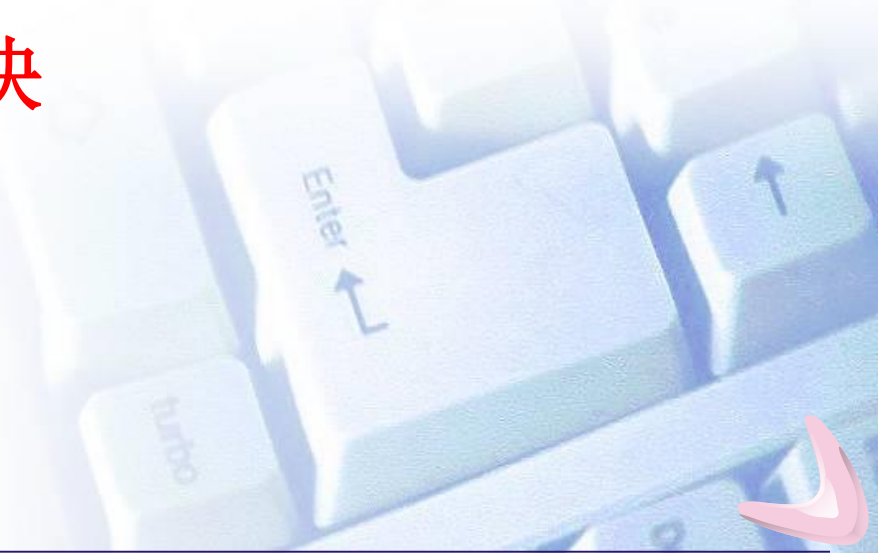
```
Void MergeList_L(LinkList La, LinkList Lb, LinkList &Lc)
{   pa=La->next;
    pb=Lb->next;
    Lc=pc=La;
    while (pa && pb)
       {  if (pa->data <= pb->data)
                { pc->next=pa;  pc=pa; pa=pa->next; }
          else
                {pc->next=pb;  pc=pb;  pb=pb->next; }
       }
    pc->next=pa?pa:pb;
    free(Lb);
}//MergeList_L
```

Click



特 点

- 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
- 插入、删除操作的速度快



线性表的链式存储结构

优缺点

优点

插入删除移动元素少

缺点

不能随机存取元素

需额外的指针空间

本章内容

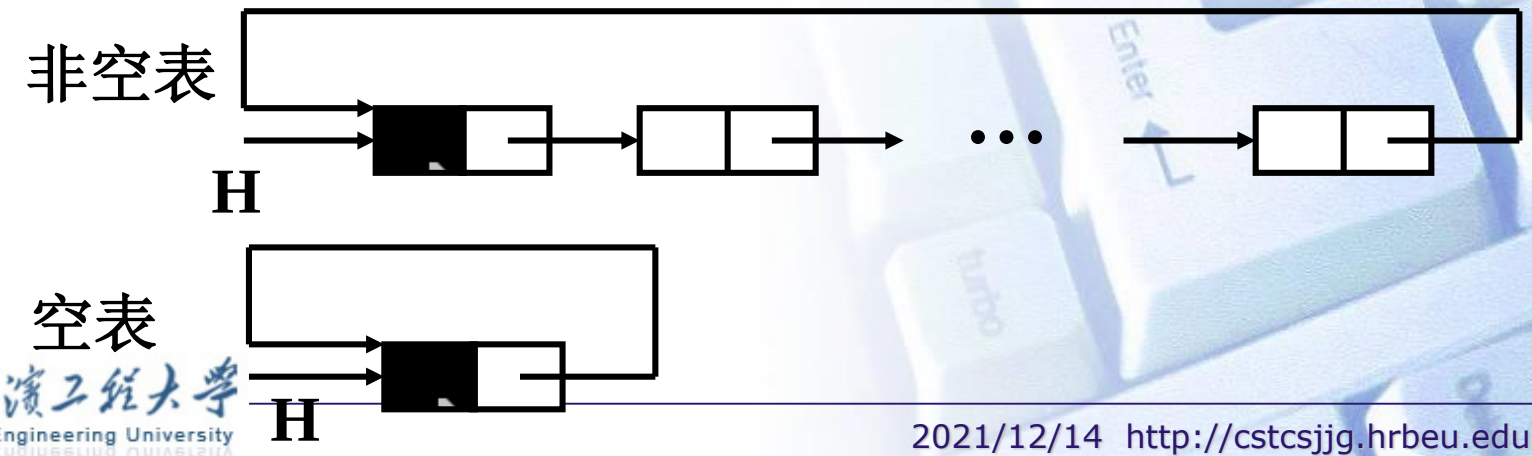
1. 线性表的类型定义
2. 线性表的顺序表示和实现
3. 线性表的链式表示和实现
4. 循环链表和双向链表
5. 一元多项式的表示及相加

循环链表 表中最后一个结点的指针指向头结点，使链表构成环状。

特　点 从表中任一结点出发均可找到表中其他结点，提高查找效率。

操　作 与单链表基本一致，判表尾条件不同

- 单链表p或p->next= =NULL
- 循环链表p或p->next= =H或L



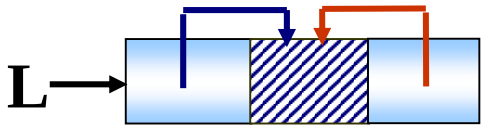
双向链表

在双向链表的结点中有两个指针域，分别指向前驱和后继。双向链表也可以有循环链表。

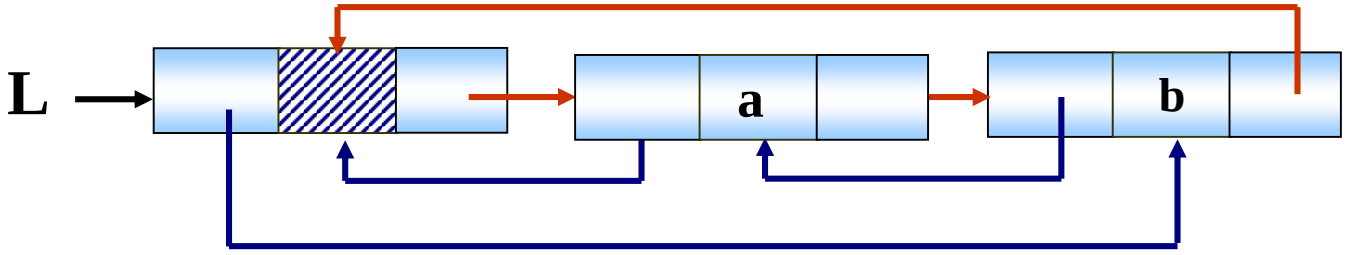
存储结构

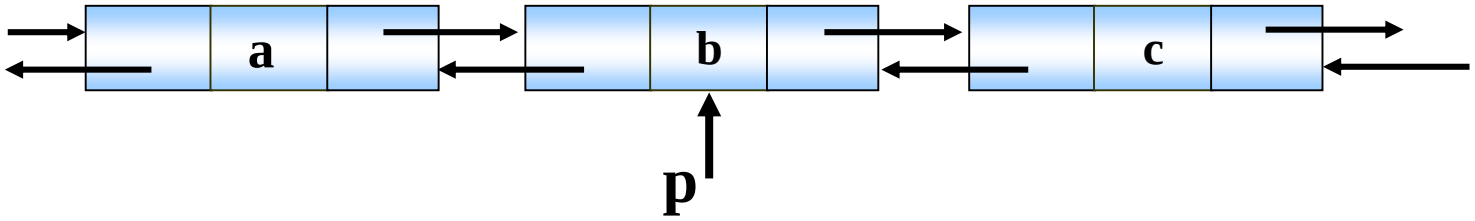

```
typedef struct DuLNode {
    ElemType      data;
    struct DuLNode  *prior;
    struct DuLNode  *next;
} DuLNode, *DuLinkList;
```

空双向循环链表



非空双向循环链表





p->prior->next= p = p->next->proir;

插入操作 在表L中第i个位置之前插入元素e

Click

- 首先在链表中找第i个元素，如果找到，则p指向它
  否则，p为空，返回错误信息
- 为元素e申请结点，插入

时间复杂度：查找O(n)，插入O(1)

删除操作 删除表L中第i个元素，赋于e

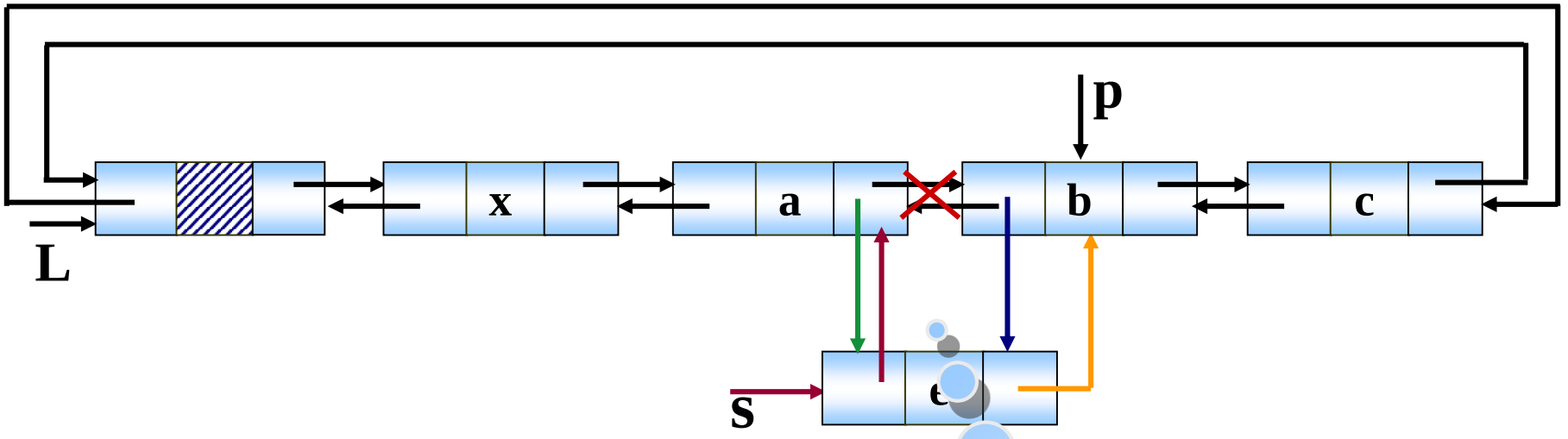
Click

- 首先在链表中找第i个元素，如果找到，则p指向它
  否则，p为空，返回错误信息
- 将p所指元素值赋给e
- 释放p所指结点

时间复杂度：查找O(n)，删除O(1)

next

在表L中第i个位置之前插入元素e



设p指针，p=GetElemP-DuL(L,i)，使其指向第i个结点

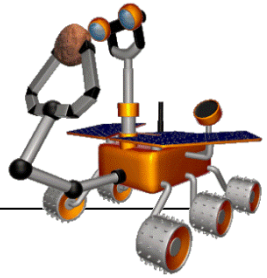
若p=null，则i不存在；p<>null，[illegible], e赋值插入

思考：四个指针的修改顺序可否任意改变？

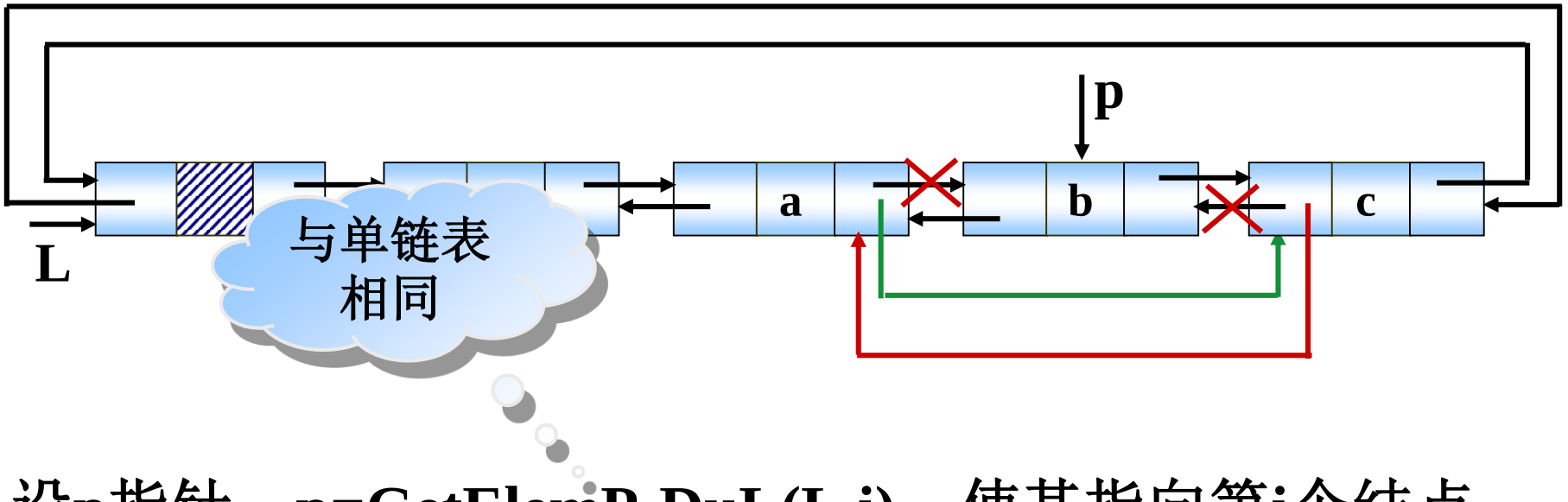
- s->prior=p->prior
- s->next=p
- p->prior->next=s
- ***p->prior=s***

双向循环链表插入算法2.18

```
Status ListInsert_DuL(DuLinkList &L, int i, ElemType e)
{     //在带头结点的双循环链线性表L中第i个位置之前
      //插入元素e,1=<i=<表长+1
    if (!(p=GetElemP_DuL(L, i)))
        return ERROR;
    if (!(s=(DuLinkList*)malloc(sizeof(DuLNode))))
          return ERROR;
    s->data=e;
    s->prior=p->prior;
    s->prior->next=s;
    s->next=p;
    p->prior=s;
    return OK;
}//ListInsert_DuL
```



在表L中删除第i个位置



设p指针，p=GetElemP-DuL(L,i)，使其指向第i个结点

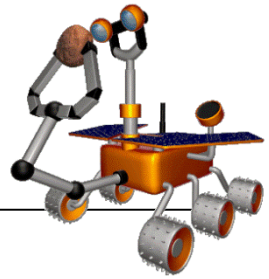
若p=null，则i不存在；p<>null，则p指向第i个结点，将其赋给e

删除

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

双向循环链表删除算法2.19

```
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)))
            return ERROR;
      e=p->data;
      p->prior->next=p->next;
      p->next->prior=p->prior;
      free(p);
      return OK;
}//ListDelete_DuL
```



操作特点

- 双指针使得链表的双向查找更为方便、快捷
- NextElem和PriorElem的执行时间为O(1)
- 涉及一个方向的指针的操作和单链表的操作相同
- 插入和删除需同时修改两个方向的指针

一元多项式的表示及相加

一元多项式

$$P_n(x) = P_0 + P_1x + P_2x^2 + \cdots\cdots + P_nx^n$$

系数表示

$$P = (P_0, P_1, P_2, \cdots\cdots, P_n)$$

特殊情况

$$S(x) = 1 + 3x^{1000} + 2x^{20000}$$ 浪费存储空间

一　般

$$P_n(x) = P_1x^{e1} + P_2x^{e2} + \cdots\cdots + P_mx^{em}$$

$0 \le e1 \le e2 \cdots\cdots \le em$（$P_i$为非零系数）

线性表表示

$$((P_1, e1),(P_2, e2),\cdots\cdots(P_m, em))$$

存储结构

可以用顺序存储结构，也可以用单链表

一元多项式的表示及相加

单链表定义

```
Typedef  struct  Pnode {
        float  coef;
        int  exp;
        struct  Pnode  *next;
}term, ElemType;
```

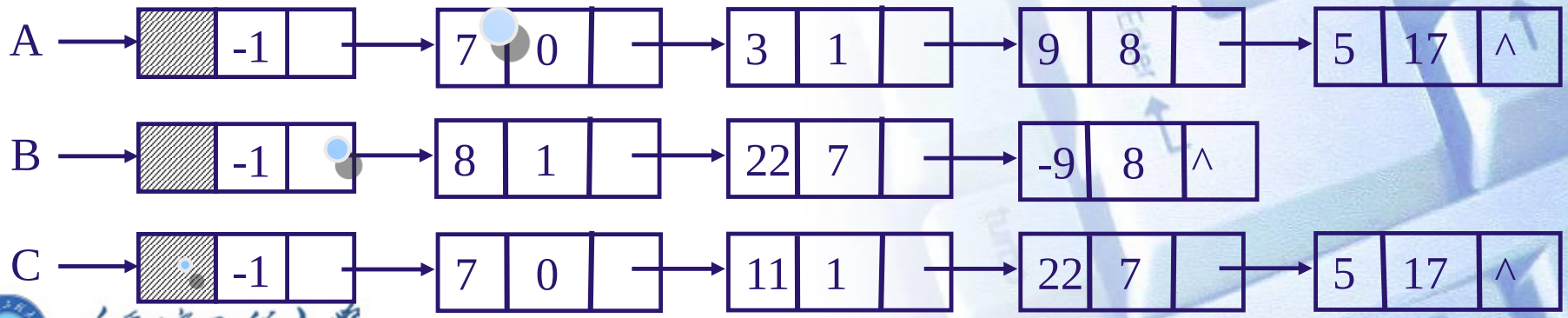
例 如

$A(x) = 7 +$

$B(x) = 8$

不需新结点，和存放在A链中

$$C(x) = A(x) + B(x) = 7 + 11x + 22x^{7} + 5x^{17}$$



一元多项式的表示及相加

算法思想

- 设p, q分别指向A,B第一结点
- 逐一将p,q分别指向结点比较，运算规则

比较 p->exp与q->exp：

- p->exp < q->exp：p结点是和多项式中的一项，p后移,q不动
- p->exp > q->exp：q结点是和多项式中的一项，将q插在p之前,q后移,p不动
- p->exp = q->exp：系数相加
  - 0：从A表中删去p，释放p,q，p,q后移
  - ≠0：修改p系数域，释放q，p,q后移

直到p或q为NULL：

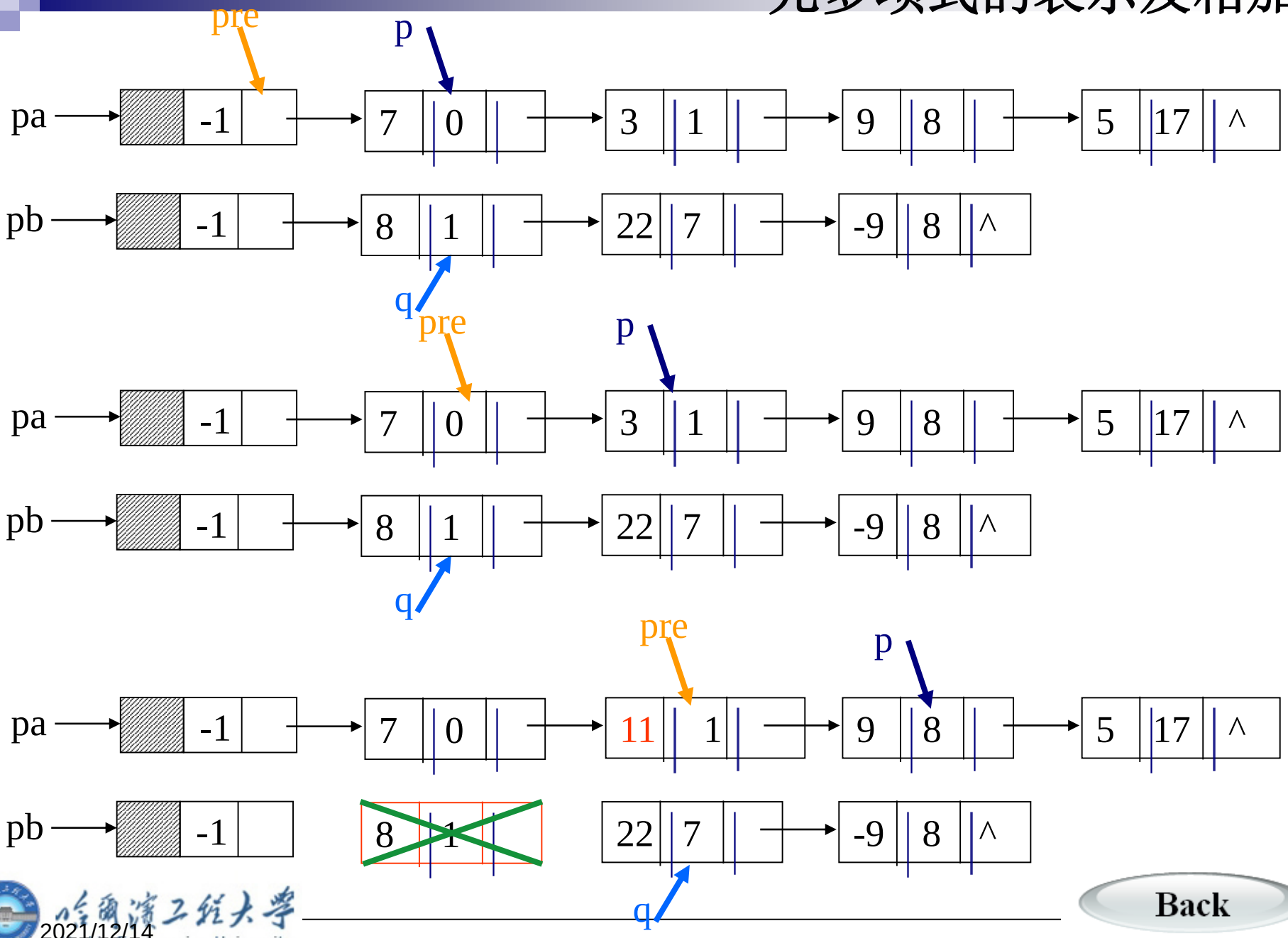
- 若q==NULL，结束
- 若p==NULL，将B中剩余部分连到A上即可

- 释放B的表头

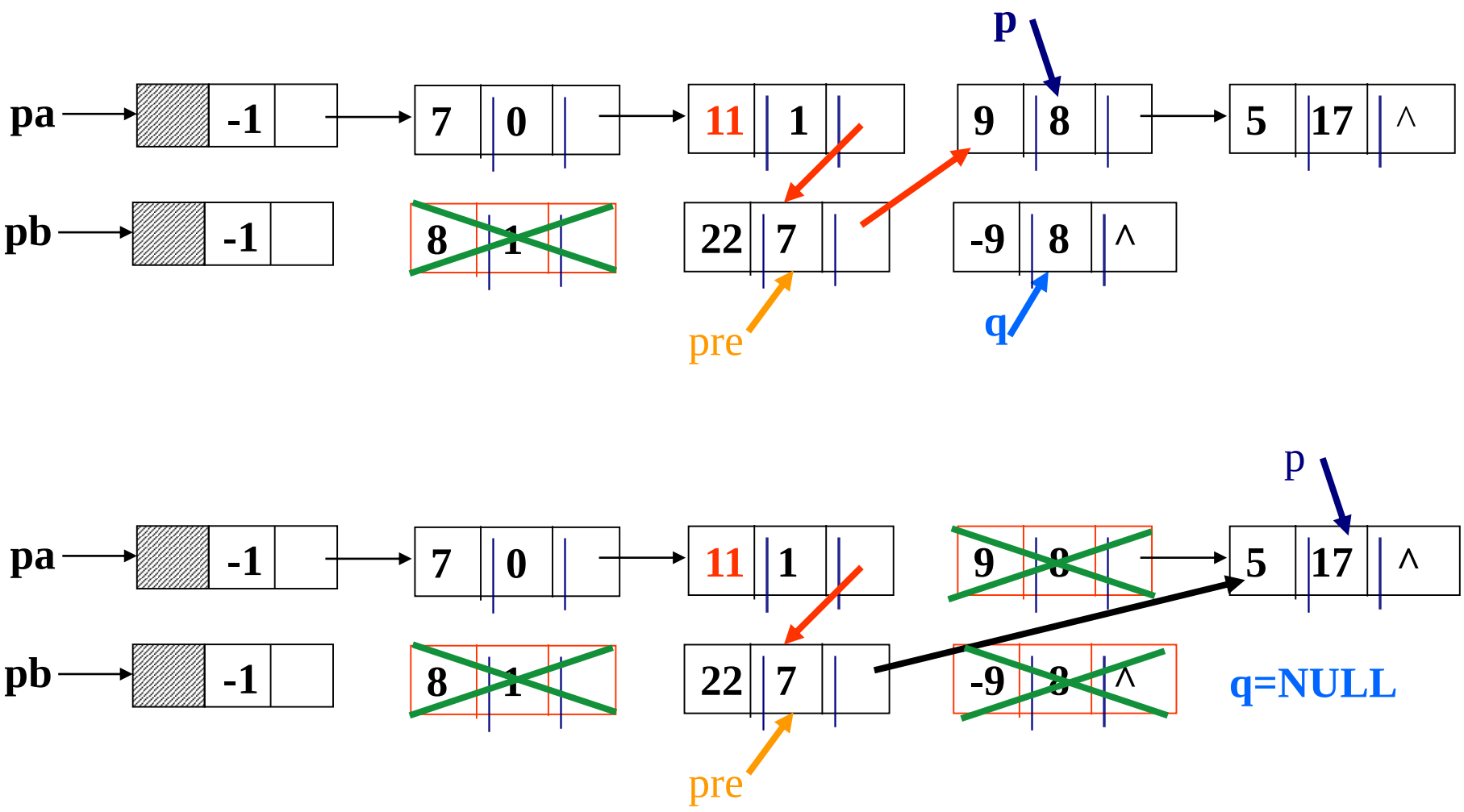
Click

next

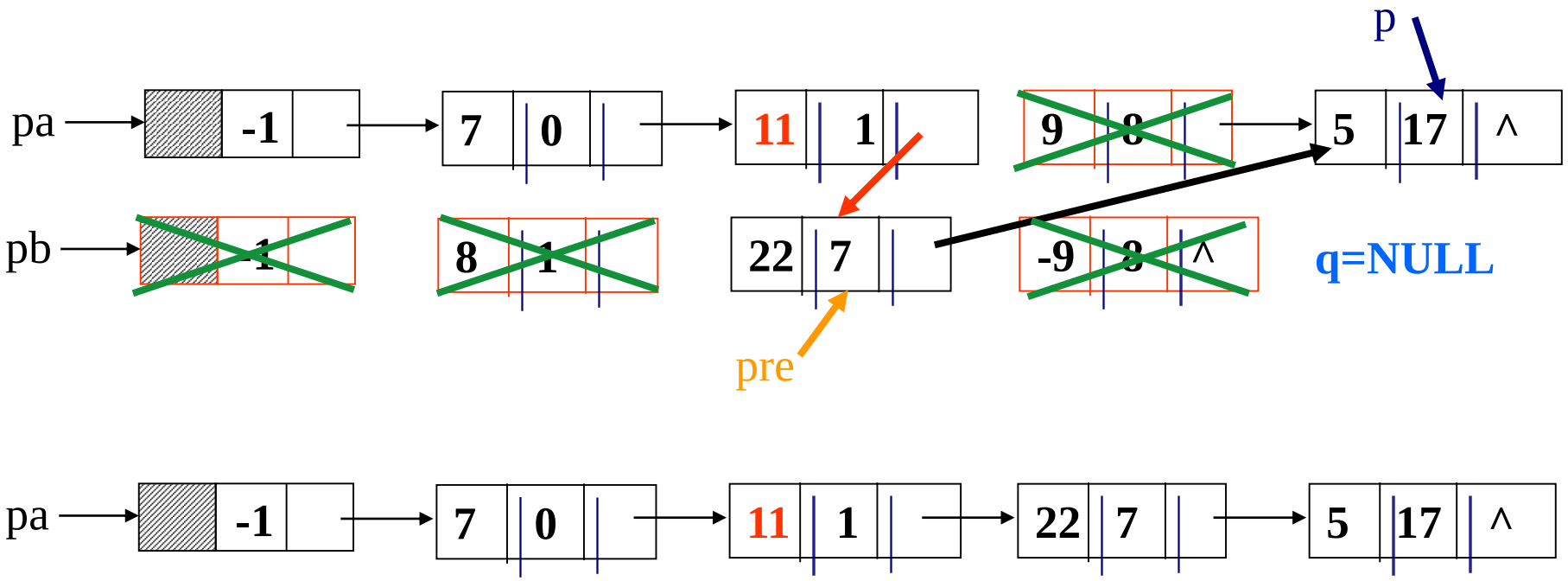
一元多项式的表示及相加



一元多项式的表示及相加



一元多项式的表示及相加



一元多项式的表示及相加算法2.20

```
void  Add_poly(polynomial  &pa, polynomial &pb)
{    p=pa->next;    q=pb->next;    pre=pa;
     while((p!=NULL) && (q!=NULL))
    {  if(p->exp<q->exp)   {  pre=p;  p=p->next;}
       else if(p->exp==q->exp)
              {
                 x=p->coef+q->coef;
                 if(x!=0)   { p->coef=x;  pre=p;}
                 else   { pre->next=p->next;  free(p);}
                 p=pre->next;
                 u=q;   q=q->next;   free(u);  //相等情况公共操作
               }//
            else
               {
                 u=q->next;  q->next=p;  pre->next=q; pre=q;  q=u;  //插入Pb节点
               }
     }
      if(q!=NULL)   pre->next=q;
      FreeNode(pb);
}//Add_poly
```

本章小结

- 线性表是n(n≥0)个数据元素的序列，通常写成

$$(a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots a_n)$$

- 线性表中除了第一个和最后一个元素之外，都只有一个前驱和一个后继。
- 线性表中每个元素都有自己确定的位置，即“位序”。
- n=0时的线性表称为“空表”，在设计和编写线性表的操作算法时，一定要考虑该算法对空表的情况的处理是否正确。

本章小结

- 顺序表
  - ❖是线性表的顺序存储结构的一种别称。
  - ❖特点是以“存储位置相邻”表示两个元素之间的前驱、后继关系。
  - ❖优点是可以随机存取表中任意一个元素。
  - ❖缺点是每作一次插入或删除操作时，平均来说必须移动表中一半元素。
  - ❖常应用于主要是为查询而很少作插入和删除操作，表长变化不大的线性表。

本章小结

- 链表
  - 是线性表的链式存储结构的别称。
  - 特点：是以“指针”指示后继元素，因此线性表的元素可以存储在存储器中任意一组存储单元中。
  - 优点：是便于进行插入和删除操作。
  - 缺点：是不能进行随机存取，每个元素的存储位置都存放在其前驱元素的指针域中，为取得表中任意一个数据元素都必须从第一个数据元素起查询。
  - 由于它是一种动态分配的结构，结点的存储空间可以随用随取，并在删除结点时随时释放，以便系统资源更有效地被利用，这对编制大型软件非常重要。

下课休息一会！