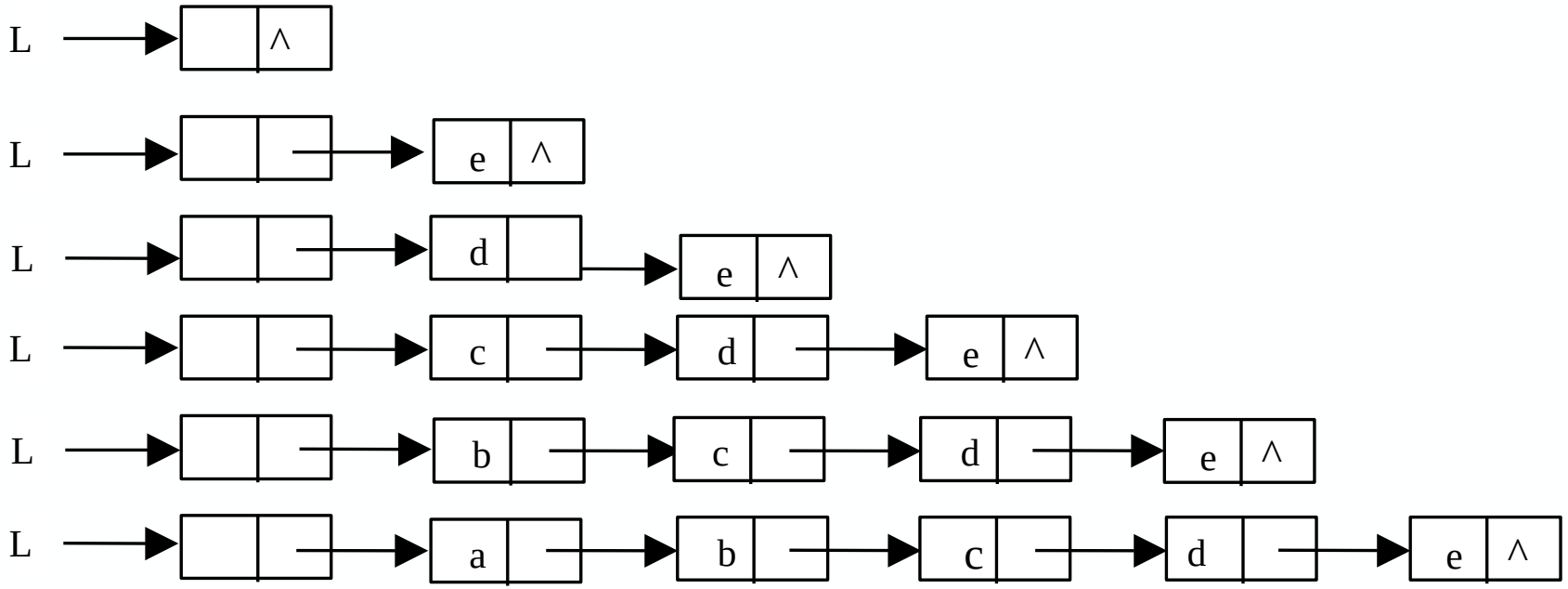
第2章 线性表

王迪

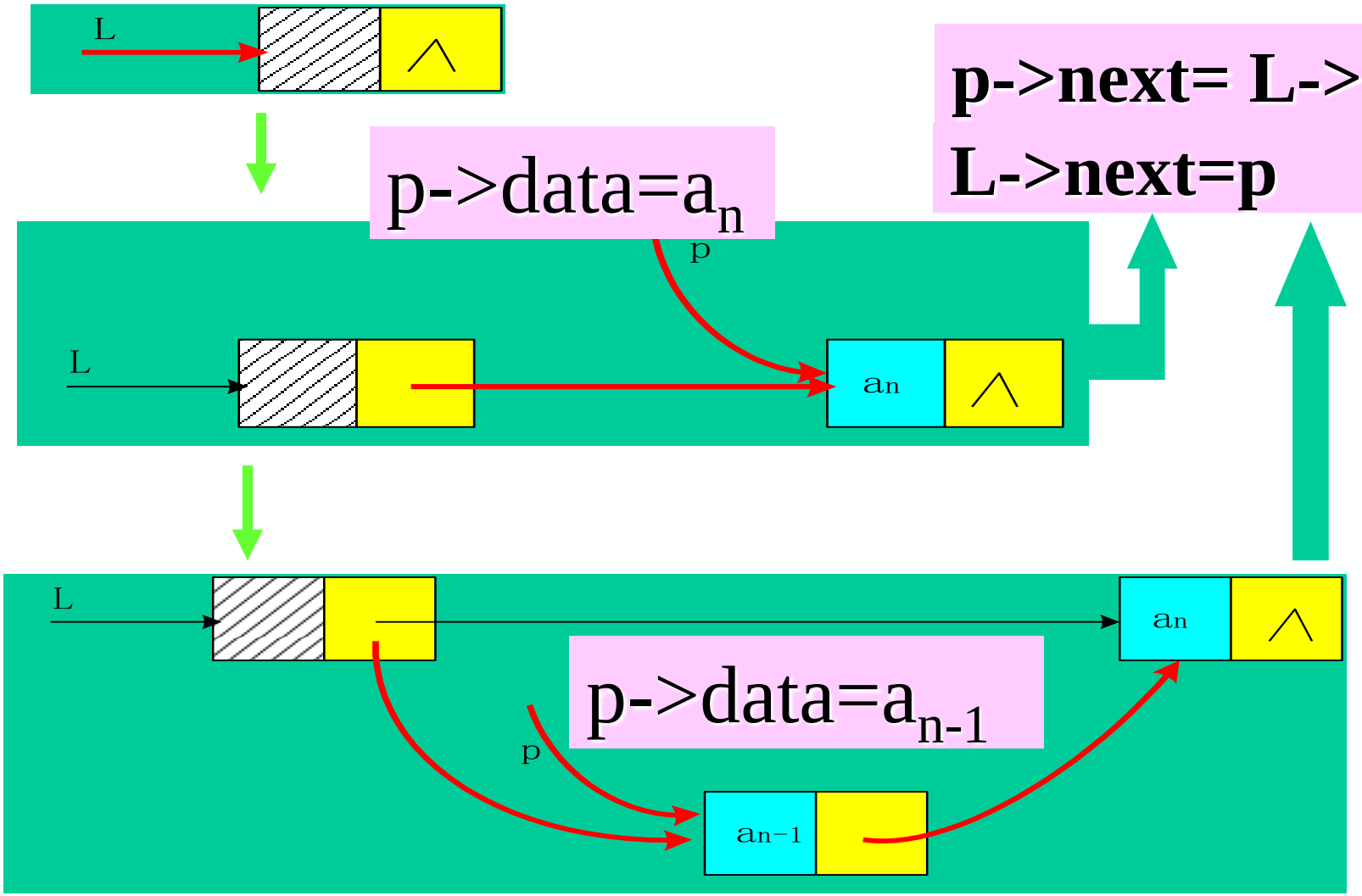
wangd@sdas.org

单链表的建立（前插法）

- 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
  - 将该新结点插入到链表的前端



单链表的建立（前插法）

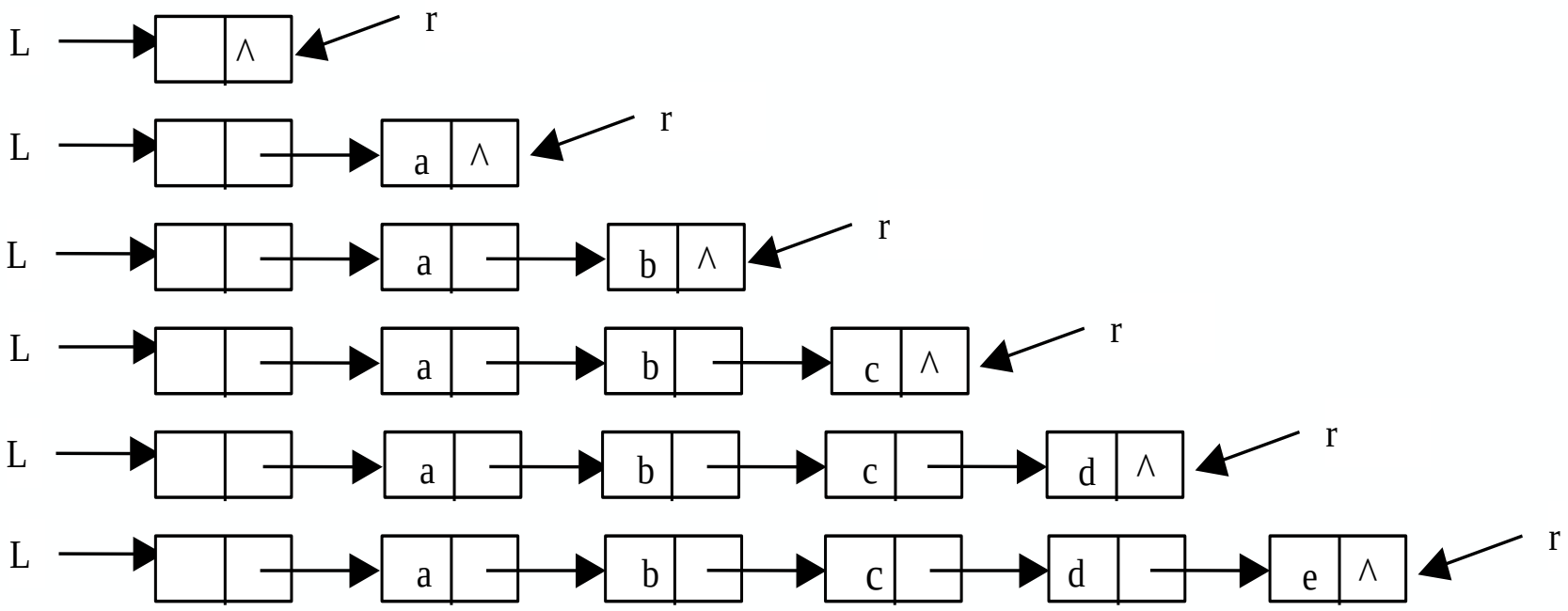


【算法描述】

```
void CreateList_F(LinkList &L,int n){
    L=new LNode;
     L->next=NULL; //先建立一个带头结点的单链表
     for(i=n;i>0;--i){
       p=new LNode; //生成新结点
       cin>>p->data; //输入元素值
       p->next=L->next;L->next=p;    //插入到表头
    }
}//CreateList_F
```

单链表的建立（尾插法）

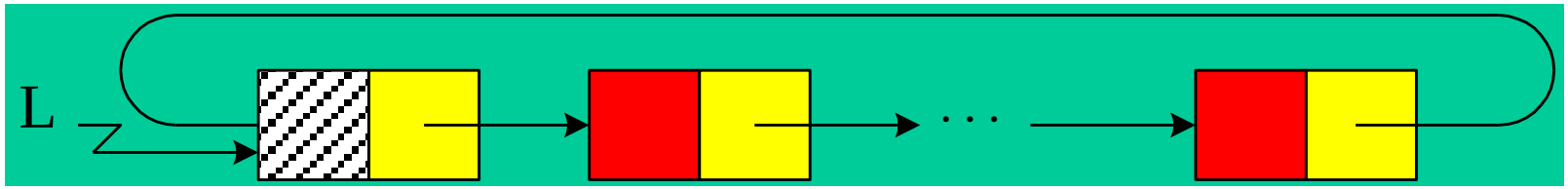
- 从一个空表L开始，将新结点逐个插入到链表的尾部，尾指针r指向链表的尾结点。
- 初始时，r同L均指向头结点。每读入一个数据元素则申请一个新结点，将新结点插入到尾结点后，r指向新结点。



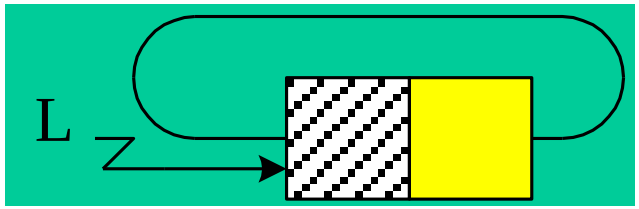
【算法描述】

```
void CreateList_L(LinkList &L,int n){
    //正位序输入n个元素的值，建立带表头结点的单链表L
    L=new LNode;
    L->next=NULL;
    r=L;        //尾指针r指向头结点
    for(i=0;i<n;++i){
      p=new LNode;              //生成新结点
      cin>>p->data;             //输入元素值
      p->next=NULL; r->next=p;              //插入到表尾
      r=p;       //r指向新的尾结点
    }
}//CreateList_L
```

2.5.3 循环链表

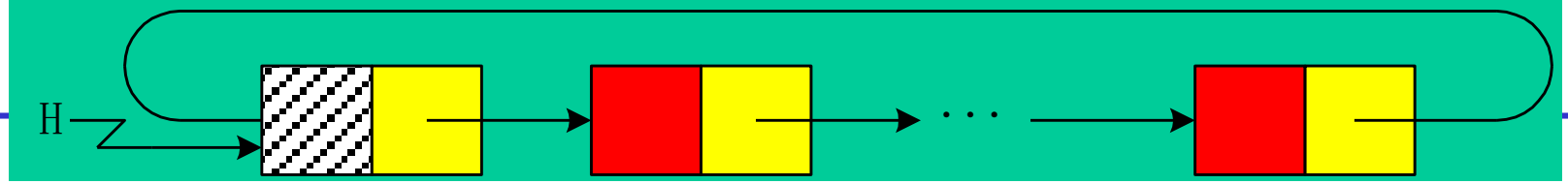


(a) 非空单循环链表



(b) 空表

L->next=L



说明

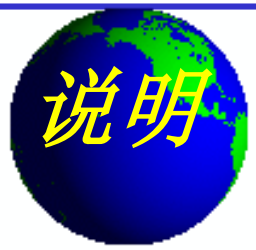
从循环链表中的任何一个结点的位置都可以找到其他所有结点，而单链表做不到；

循环链表中没有明显的尾端

如何避免死循环

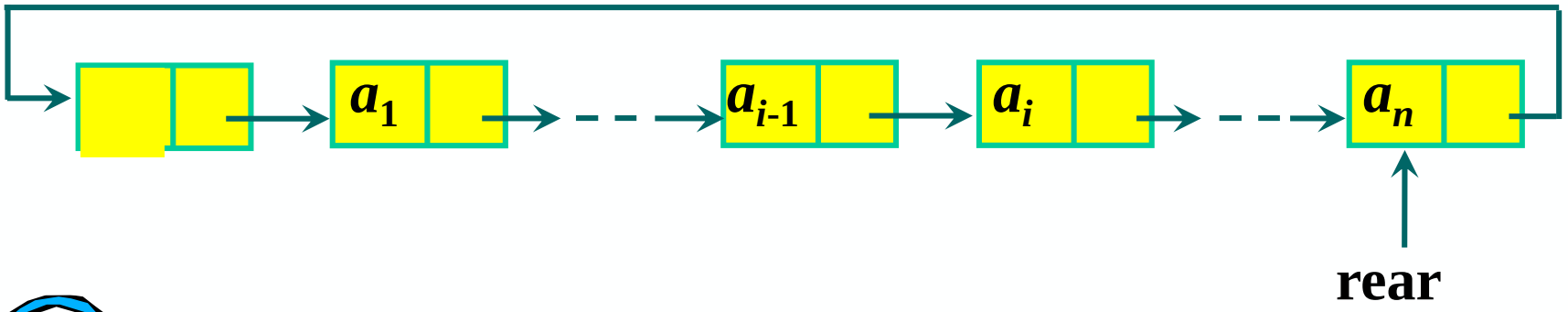
循环条件：**p!=NULL➔p!=L**

p->next!=NULL➔p->next!=L



对循环链表，有时不给出头指针，而给出尾指针

可以更方便的找到第一个和最后一个结点

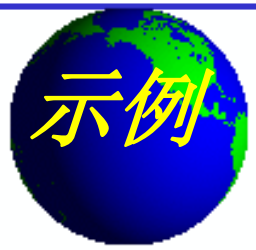


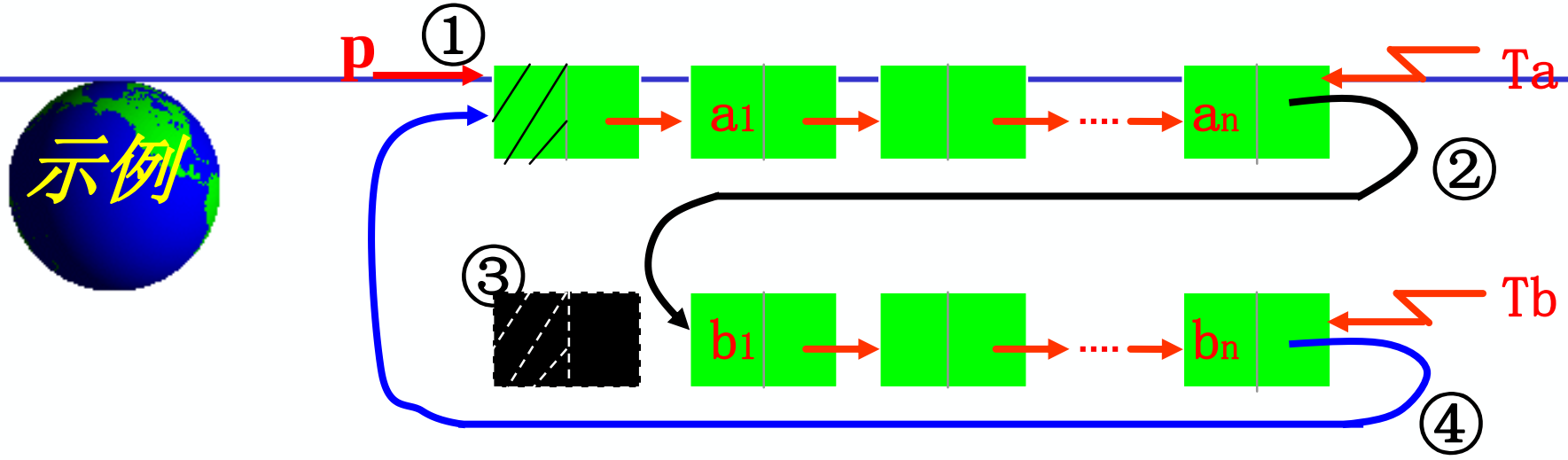
如何查找开始结点和终端结点？

开始结点：**rear->next->next**

终端结点：**rear**

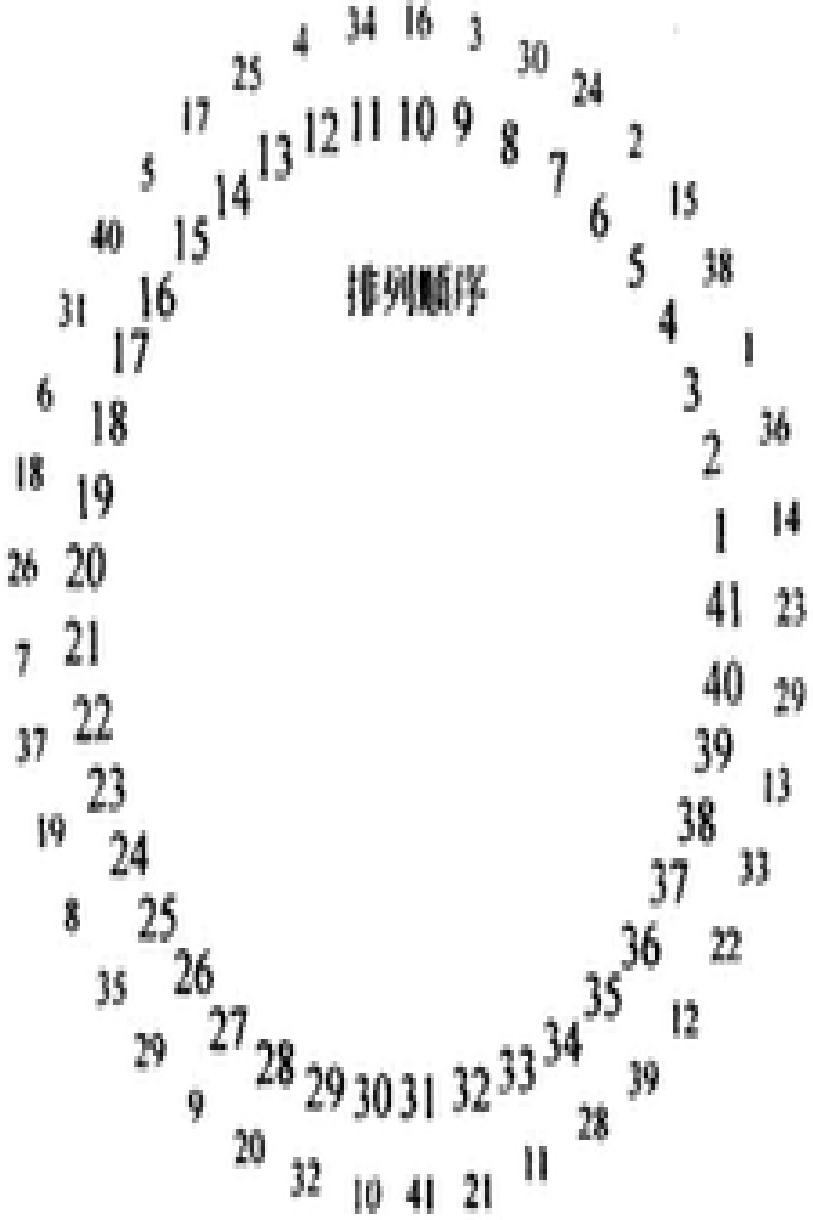
循环链表的合并






```
LinkList  Connect(LinkList  Ta,LinkList  Tb)
{//假设Ta、Tb都是非空的单循环链表
        p=Ta->next;                          //①p存表头结点
        Ta->next=Tb->next->next;//②Tb表头连结Ta表尾
        deleteTb->next;                      //③释放Tb表头结点
        Tb->next=p;                          //④修改指针
        return  Tb;
}
```

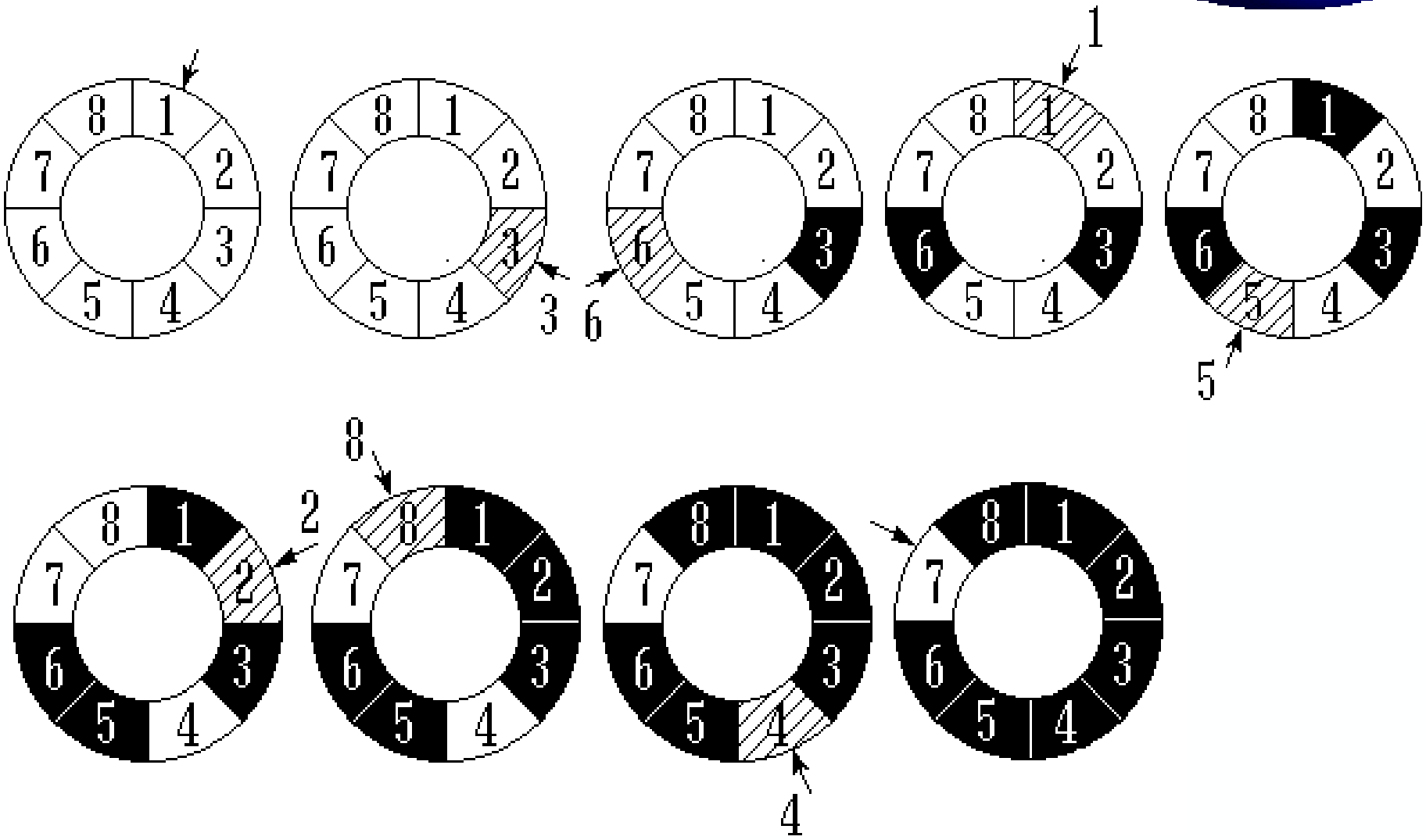
约瑟夫问题



著名犹太历史学家 **Josephus**

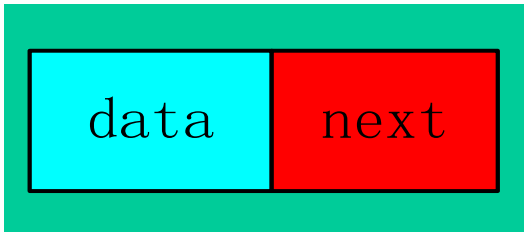
- 在罗马人占领乔塔帕特后
- 39 个犹太人与Josephus及他的朋友躲到一个洞中
- 39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式
- 41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止
- 然而Josephus 和他的朋友并不想遵从，Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏

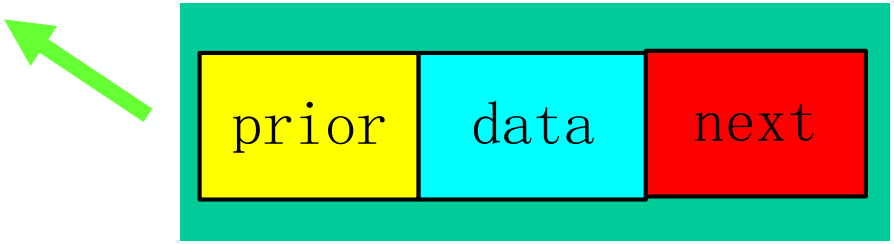
- 例如 $n = 8$ $m = 3$

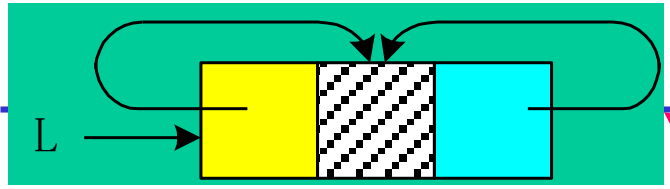


2.5.4 双向链表

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

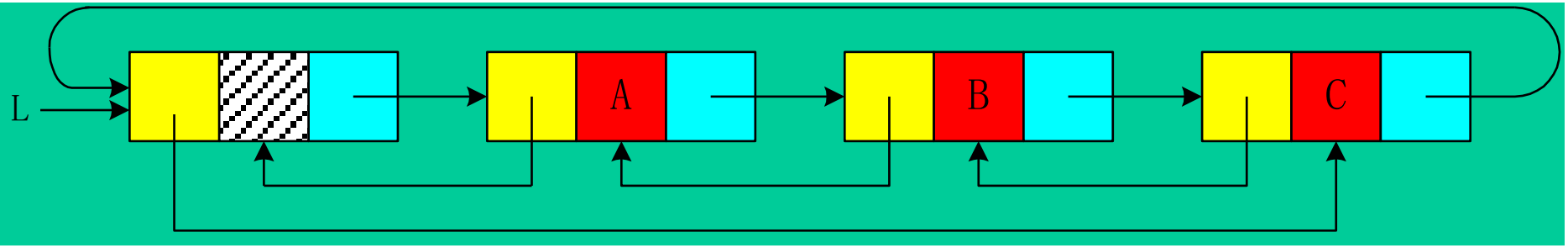






(a)　空双向循环链表

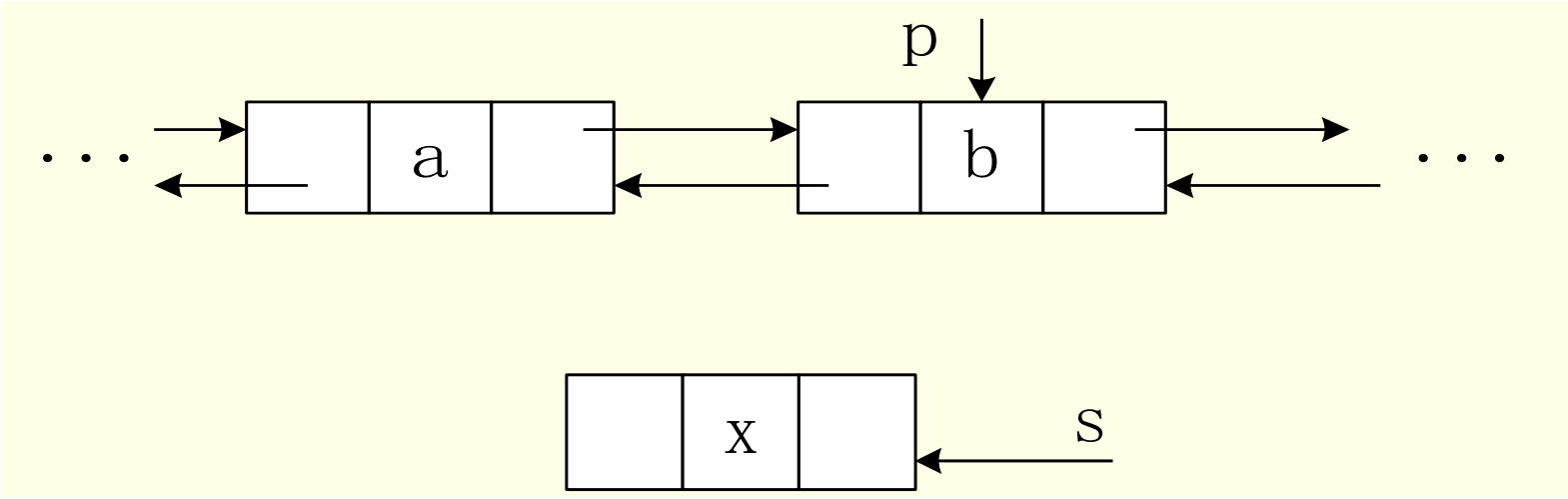
L->next=L



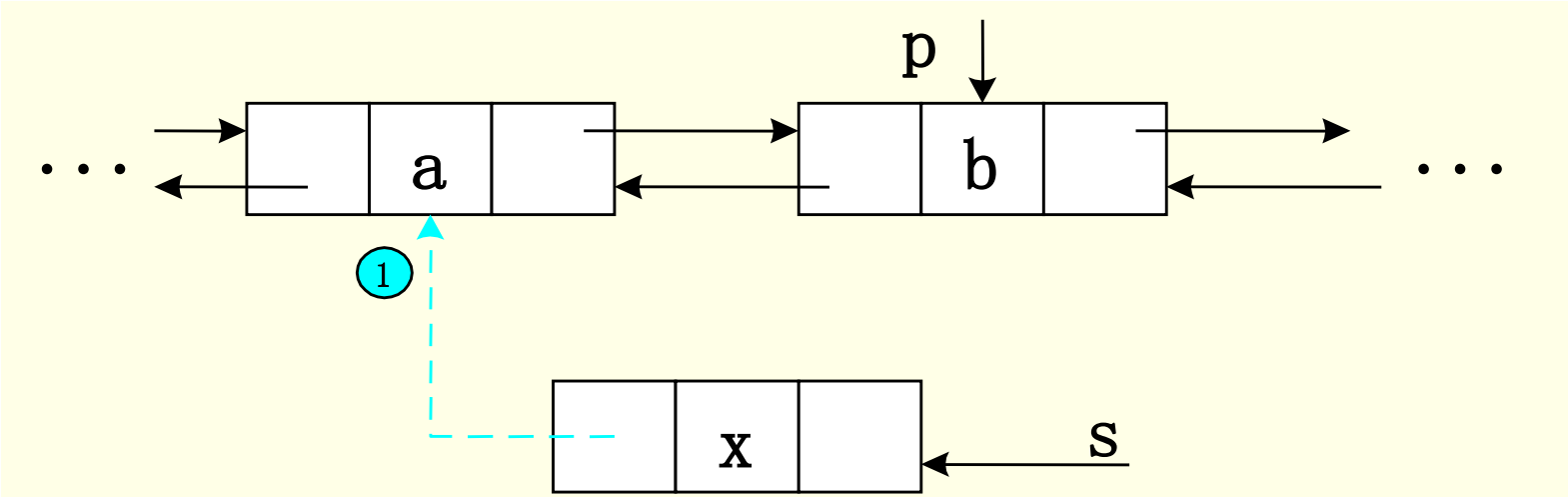
(b) 双向循环链表

d->next->prior = d->prior->next = d

双向链表的插入

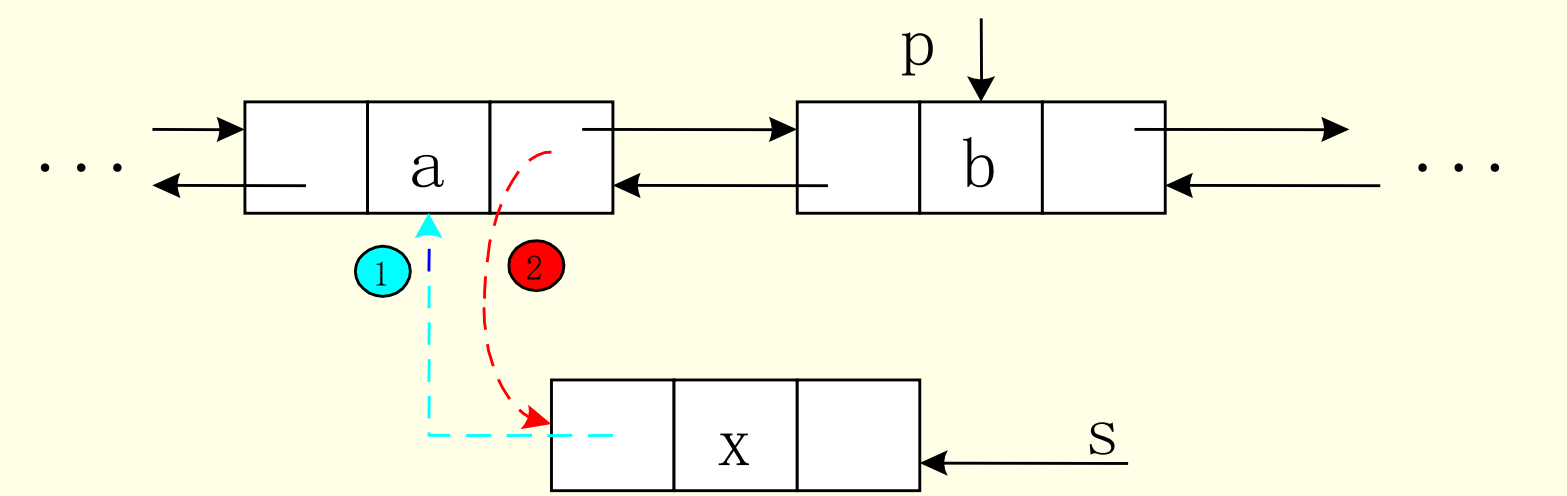


双向链表的插入



1. s->prior=p->prior;

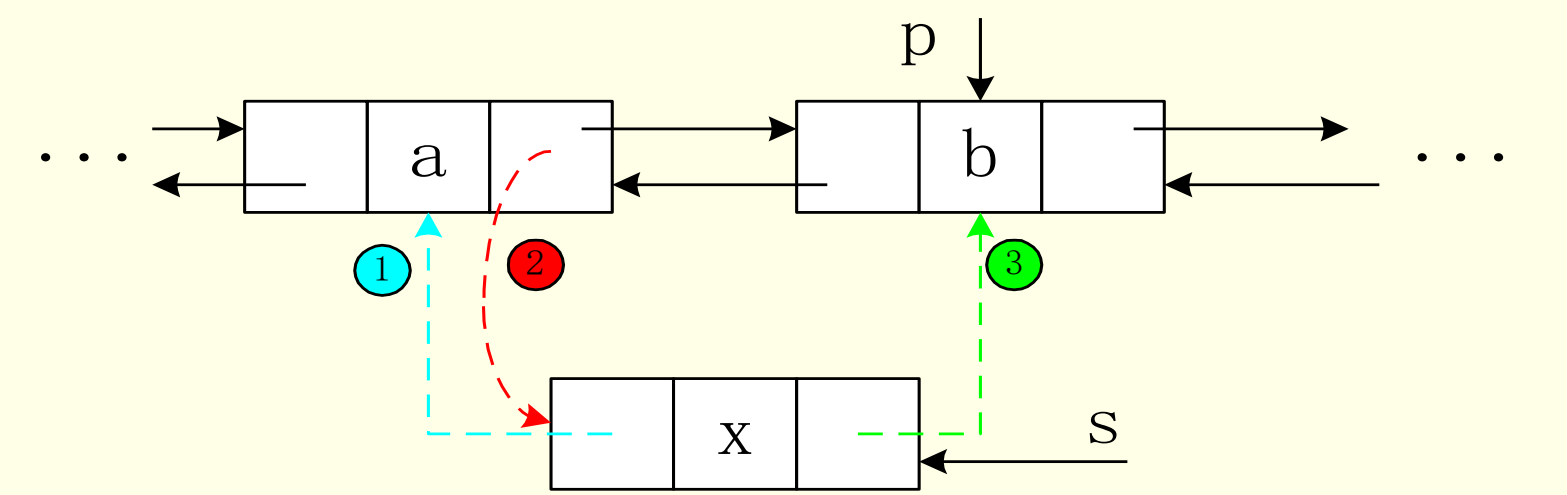
双向链表的插入



1. s->prior=p->prior;

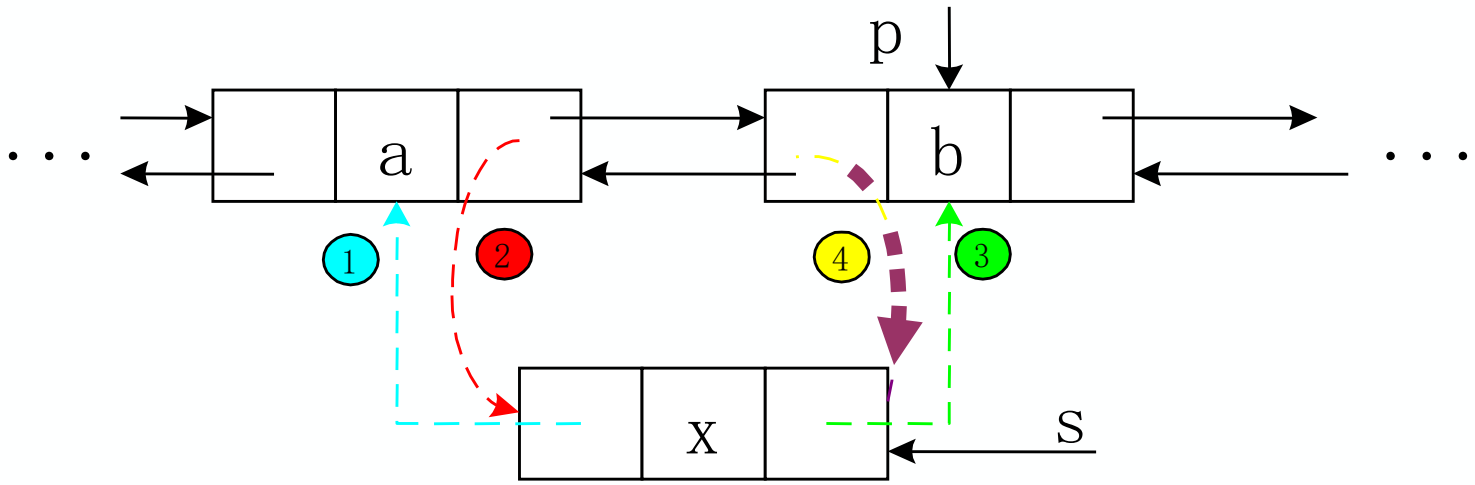
2. p->prior->next=s;

双向链表的插入



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

双向链表的插入



1. s->prior=p->prior;

2. p->prior->next=s;

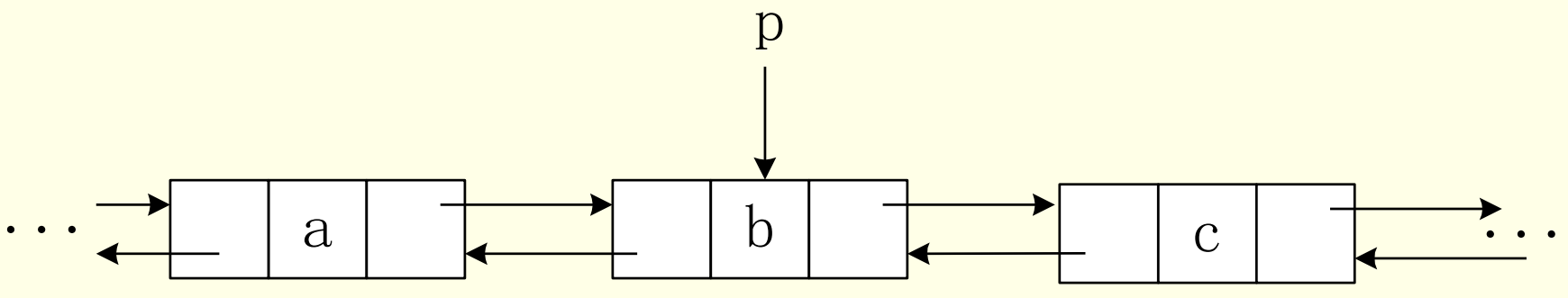
3. s->next=p;

4. p->prior=s;

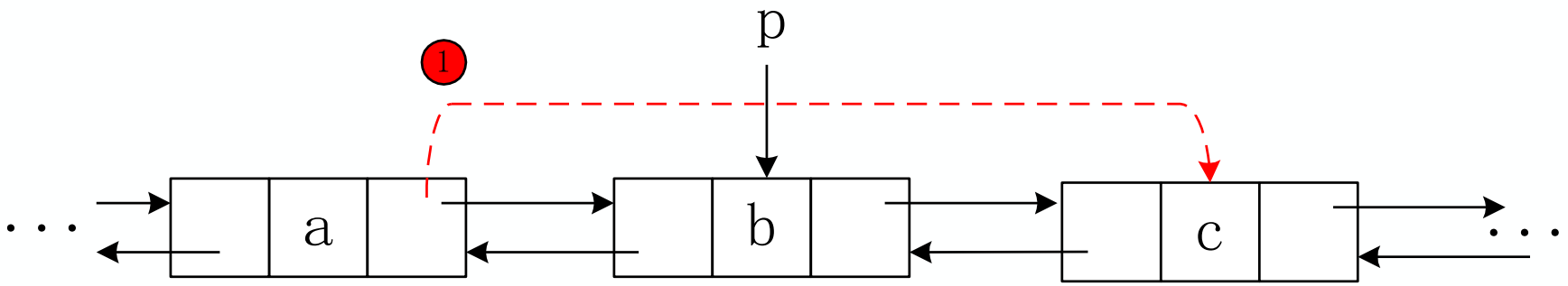
双向链表的插入

```
Status ListInsert_DuL(DuLinkList &L,int i,ElemType e){
  if(!(p=GetElemP_DuL(L,i))) return ERROR;
   s=new DuLNode;
  s->data=e;
  s->prior=p->prior;
  p->prior->next=s;
  s->next=p;
  p->prior=s;
  return OK;
}
```

双向链表的删除

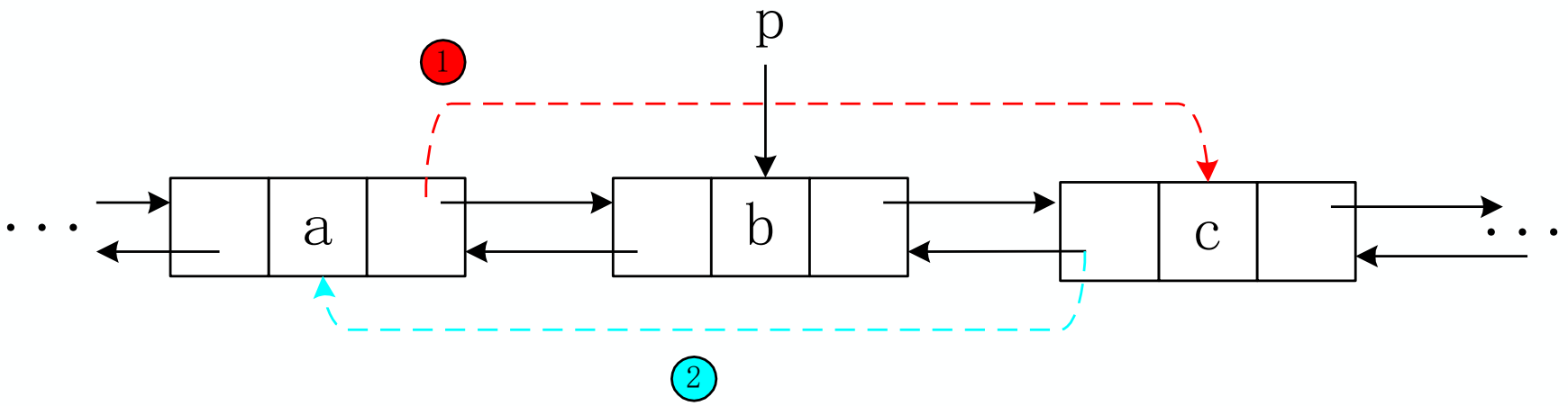


双向链表的删除



1. p->prior->next=p->next;

双向链表的删除



1. p->prior->next=p->next;

2. p->next->prior=p->prior;

双向链表的删除

```
Status ListDelete_DuL(DuLinkList &L,int i,ElemType &e){
   if(!(p=GetElemP_DuL(L,i)))     return ERROR;
   e=p->data;
   p->prior->next=p->next;
   p->next->prior=p->prior;
   delete p;
   return OK;
}
```

顺序表（顺序存储结构）的特点

（1）利用数据元素的存储位置表示线性表中相邻数据元素之间的前后关系，即线性表的逻辑结构与存储结构一致

（2）在访问线性表时，可以快速地计算出任何一个数据元素的存储地址。因此可以粗略地认为，访问每个元素所花时间相等

这种存取元素的方法被称为随机存取法

顺序表的优缺点

优点：

- ✓存储密度大（结点本身所占存储量/结点结构所占存储量）
- ✓可以随机存取表中任一元素

缺点：

- ✓在插入、删除某一元素时，需要移动大量元素
- ✓浪费存储空间
- ✓属于静态存储形式，数据元素的个数不能自由扩充

为克服这一缺点  链表

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顺序存取法

链表的优缺点

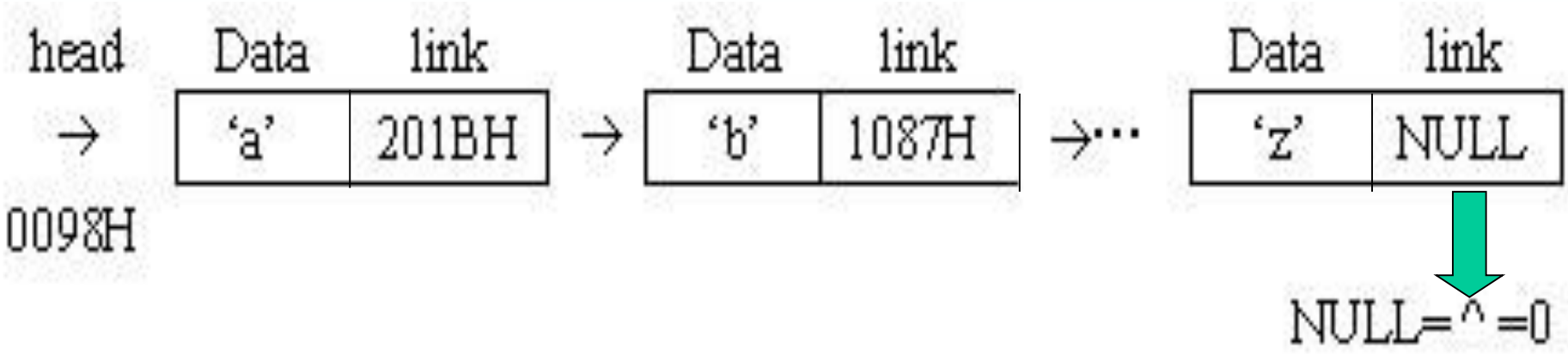
优点

- 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

链表的优缺点

缺点

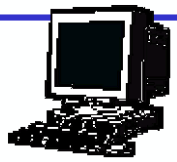
- 存储密度小
- 存取效率不高，必须采用顺序存取，即存取数据元素时，只能按链表的顺序进行访问（顺藤摸瓜）



2.6 顺序表和链表的比较

存储结构 比较项目		顺序表	链表
空间	存储空间	预先分配，会导致空间闲置或溢出现象	动态分配，不会出现存储空间闲置或溢出现象
	存储密度	不用为表示结点间的逻辑关系而增加额外的存储开销，存储密度等于1	需要借助指针来体现元素间的逻辑关系，存储密度小于1
时间	存取元素	随机存取，按位置访问元素的时间复杂度为O(1)	顺序存取，按位置访问元素时间复杂度为O(*n*)
	插入、删除	平均移动约表中一半元素，时间复杂度为O(*n*)	不需移动元素，确定插入、删除位置后，时间复杂度为O(1)
适用情况		① 表长变化不大，且能事先确定变化的范围 ② 很少进行插入或删除操作，经常按元素位置序号访问数据元素	① 长度变化较大 ② 频繁进行插入或删除操作

2.7 线性表的应用



1 线性表的合并

2 有序表的合并

2.7.1 线性表的合并

问题描述：

假设利用两个线性表La和Lb分别表示两个集合A和B，现要求一个新的集合

$$A=A\cup B$$

La=(7, 5, 3, 11)

Lb=(2, 6, 3)

La=(7, 5, 3, 11, 2, 6)

【算法步骤】

依次取出Lb 中的每个元素，执行以下操作：

在La中查找该元素

如果找不到，则将其插入La的最后

【算法描述】

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i,e);
     if(!LocateElem(La,e))
        ListInsert(&La,++La_len,e);
  }
}
```

$O(ListLength(LA) \times ListLength(LB))$

2.7.2 有序表的合并

问题描述：

已知线性表La 和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中的数据元素仍按值非递减有序排列.

La=(1 ,7, 8)

Lb=(2, 4, 6, 8, 10, 11)

Lc=(1, 2, 4, 6, 7 , 8, 8, 10, 11)

•

【算法步骤】—有序的顺序表合并

(1) 创建一个空表Lc

(2) 依次从 La 或 Lb 中“摘取”元素值较小的结点插入到 Lc 表的最后，直至其中一个表变空为止

(3) 继续将 La 或 Lb 其中一个表的剩余结点插入在 Lc 表的最后

【算法描述】—有序的顺序表合并

```
void MergeList_Sq(SqList LA,SqList LB,SqList &LC){
    pa=LA.elem;  pb=LB.elem;          //指针pa和pb的初值分别指向两个表的第一个元素
    LC.length=LA.length+LB.length;          //新表长度为待合并两表的长度之和
    LC.elem=new ElemType[LC.length];        //为合并后的新表分配一个数组空间
    pc=LC.elem;                             //指针pc指向新表的第一个元素
    pa_last=LA.elem+LA.length-1;     //指针pa_last指向LA表的最后一个元素
    pb_last=LB.elem+LB.length-1;     //指针pb_last指向LB表的最后一个元素
    while(pa<=pa_last && pb<=pb_last){       //两个表都非空

     if(*pa<=*pb) *pc++=*pa++;               //依次“摘取”两表中值较小的结点
     else *pc++=*pb++;      }
    while(pa<=pa_last)  *pc++=*pa++;         //LB表已到达表尾
    while(pb<=pb_last)  *pc++=*pb++;        //LA表已到达表尾
}//MergeList_Sq
```

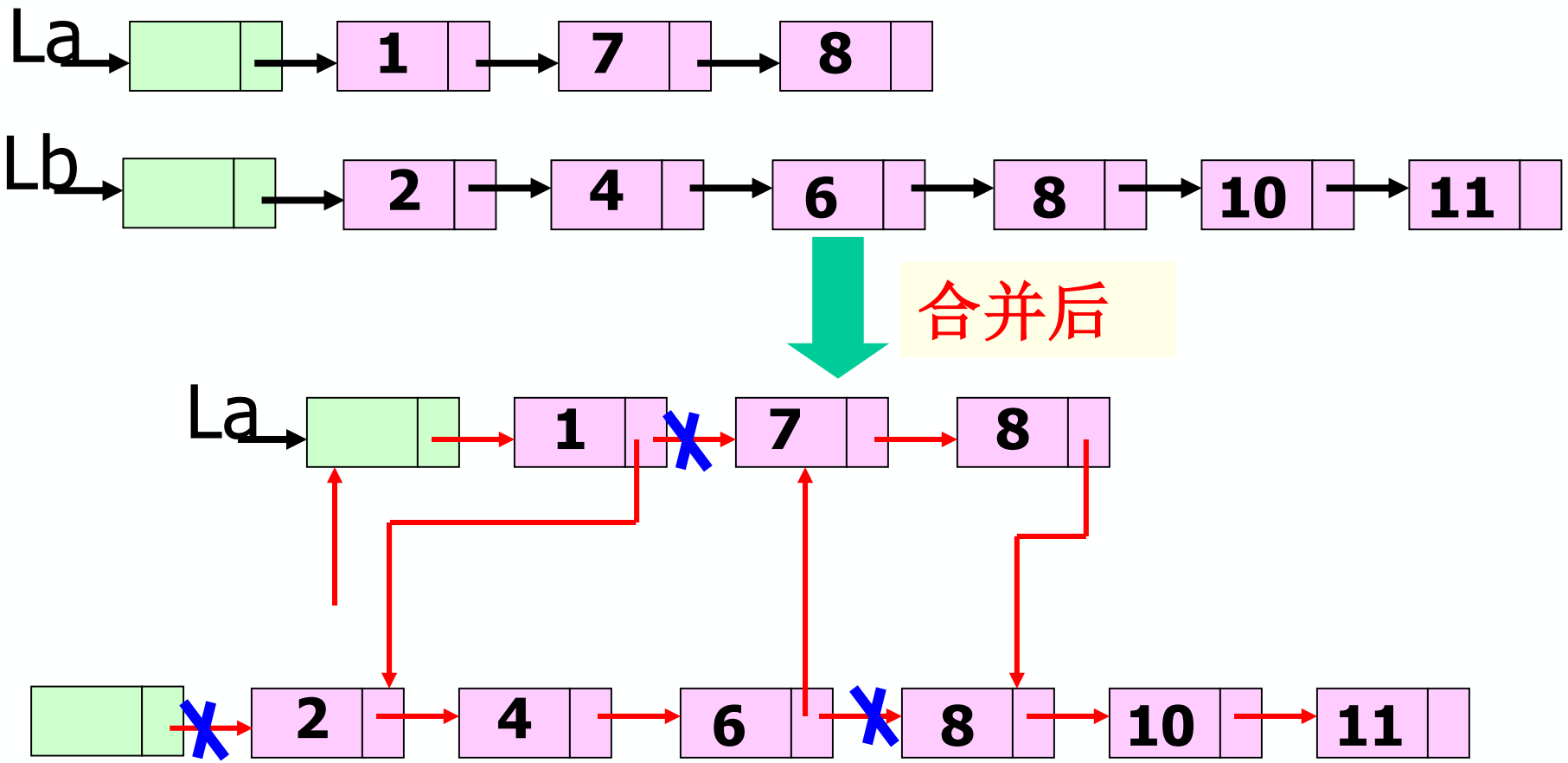
T(n)= $O(ListLength(LA)+ListLength(LB))$

S(n)= O(n)

有序链表合并——重点掌握

- ✓将这两个有序链表合并成一个有序的单链表。
- ✓要求结果链表仍使用原来两个链表的存储空间，不另外占用其它的存储空间。
- ✓表中允许有重复的数据。

有序链表合并——重点掌握



【算法步骤】—有序的链表合并

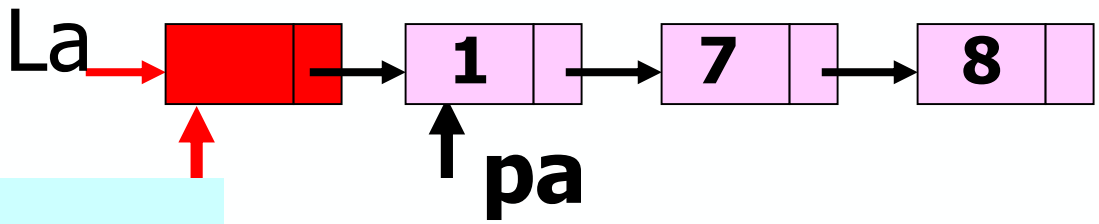
(1)Lc指向La

(2) 依次从 La 或 Lb 中“摘取”元素值较小的结点插入到 Lc 表的最后，直至其中一个表变空为止

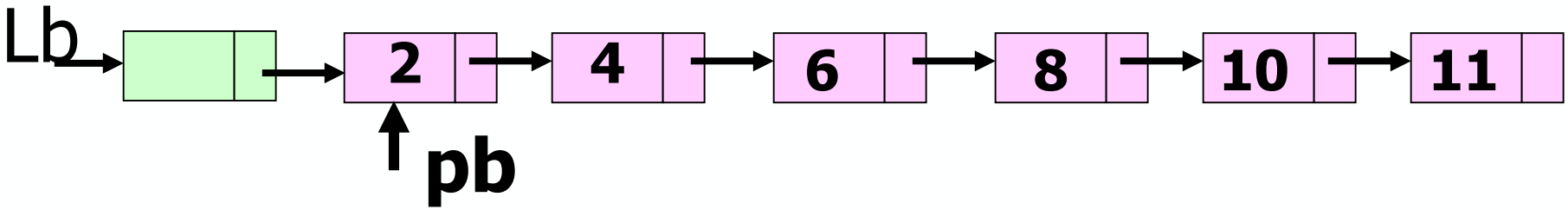
(3) 继续将 La 或 Lb 其中一个表的剩余结点插入在 Lc 表的最后

(4) 释放 Lb 表的表头结点

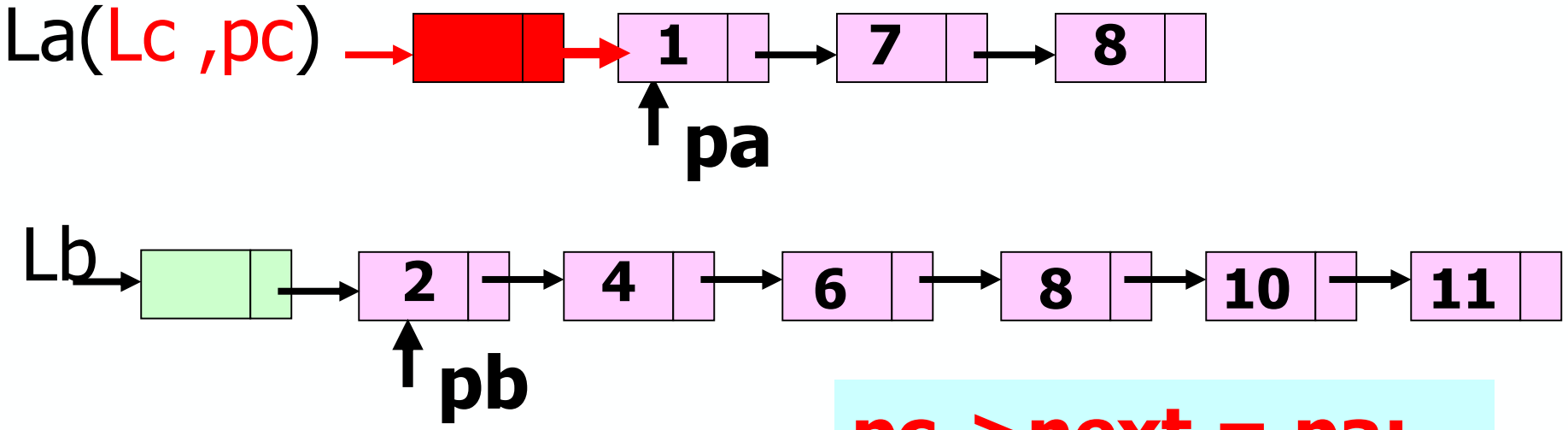
有序链表合并（初始化）



Lc = pc

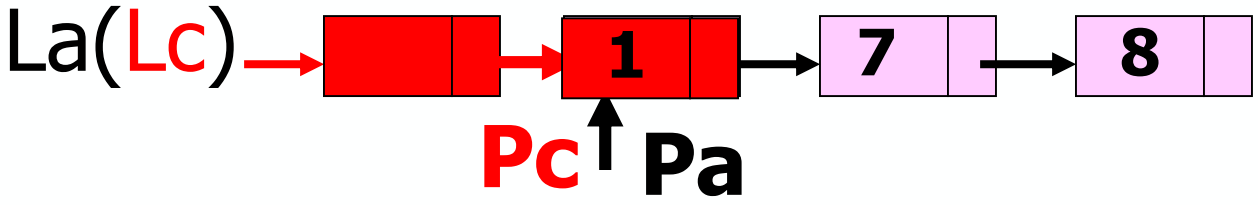


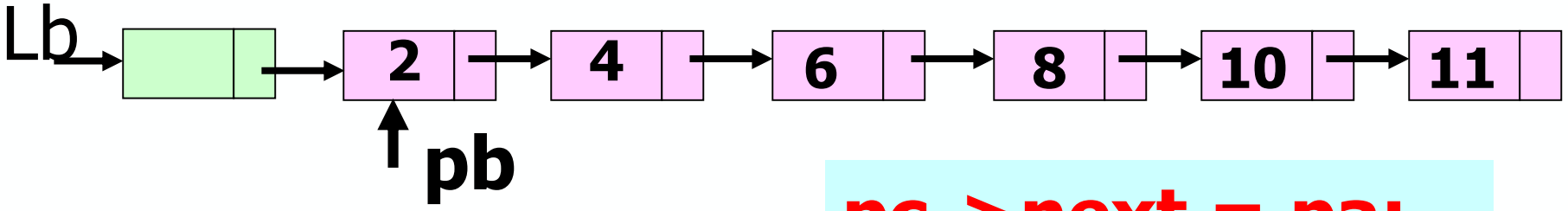
有序链表合并(pa->data < =pb->data)



pc->next = pa;

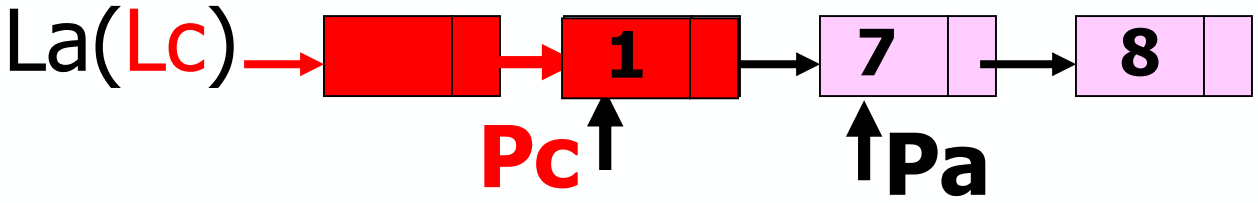
有序链表合并(pa->data < ＝pb->data)

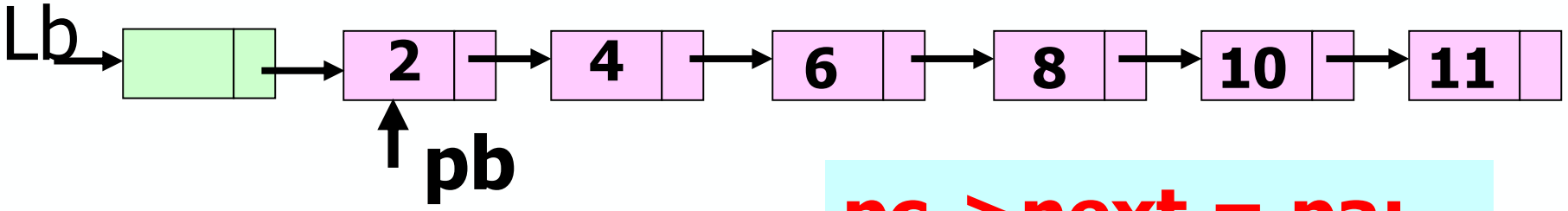





```
pc->next = pa;
pc= pa;
```

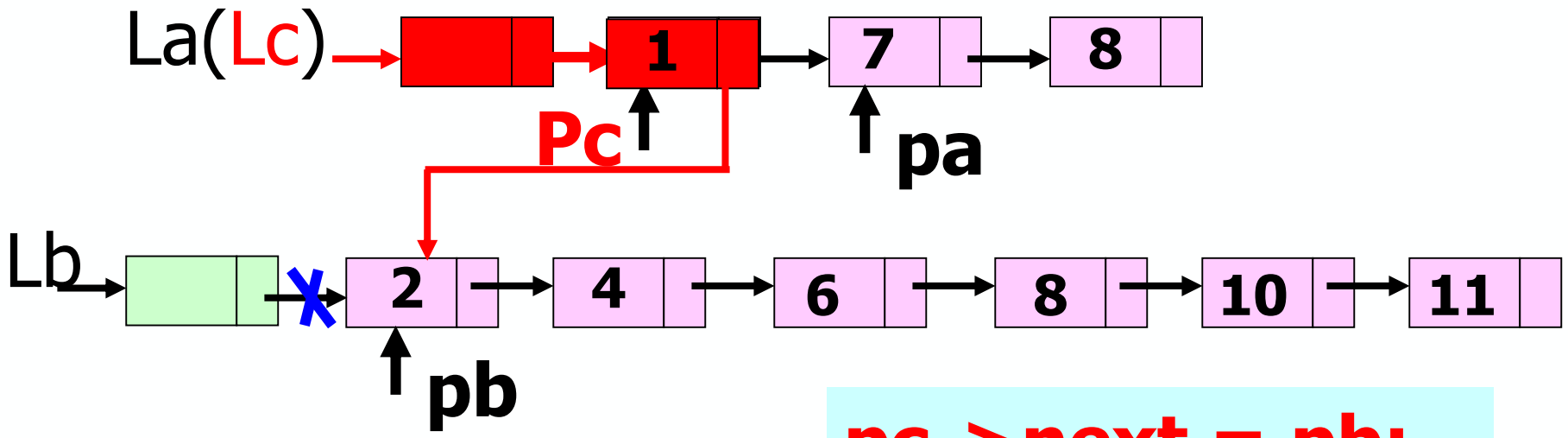
有序链表合并(pa->data < ＝pb->data)





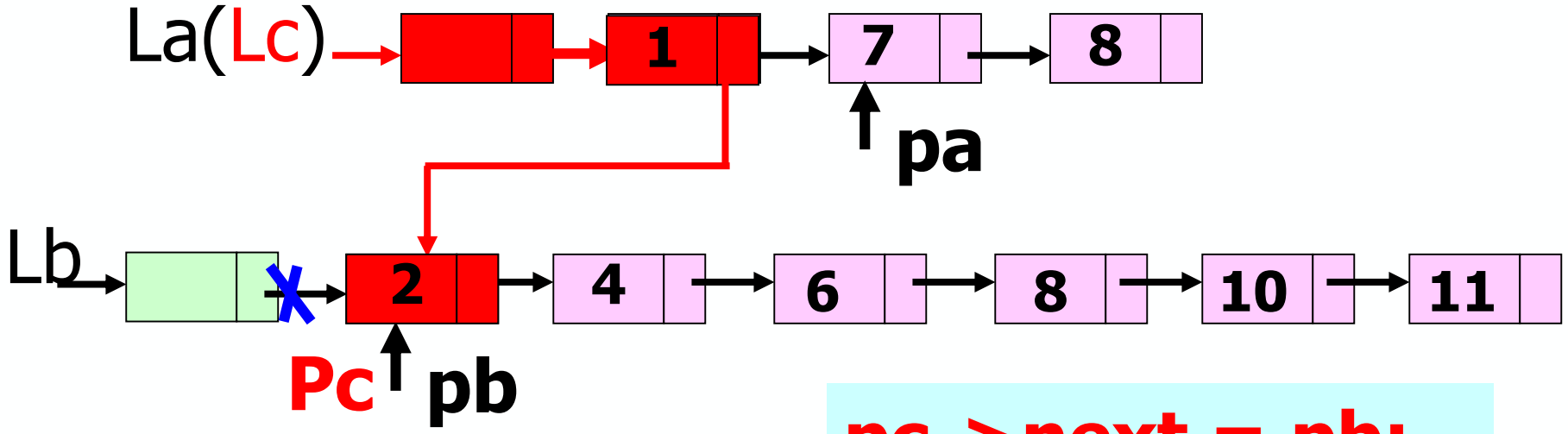

```
pc->next = pa;
pc= pa;
pa = pa->next;
```

有序链表合并(pa->data >pb->data)



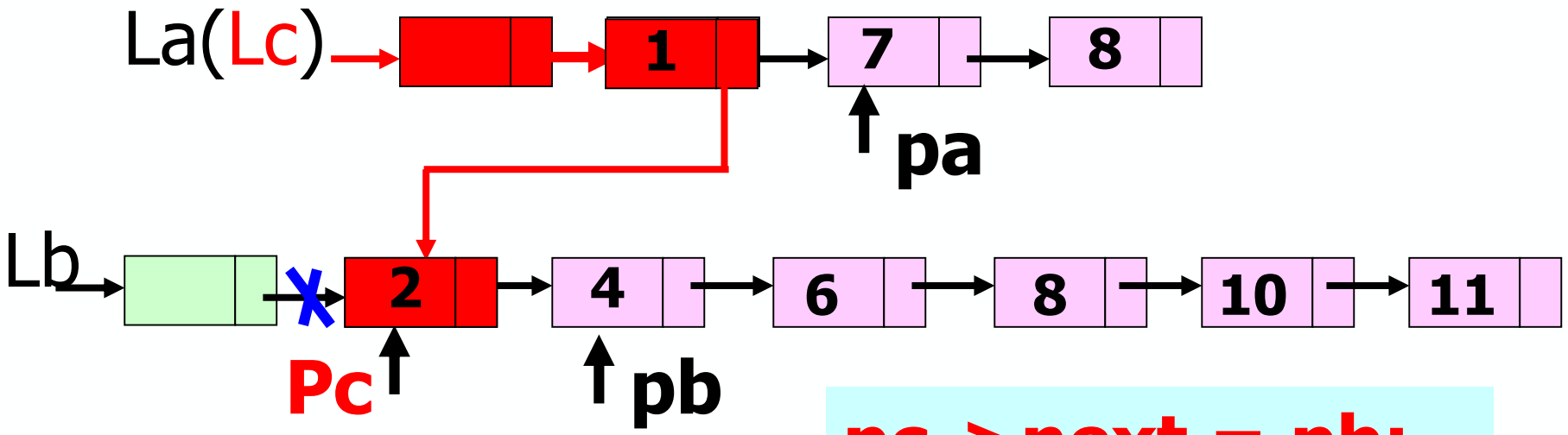
pc->next = pb;

有序链表合并(pa->data >pb->data)



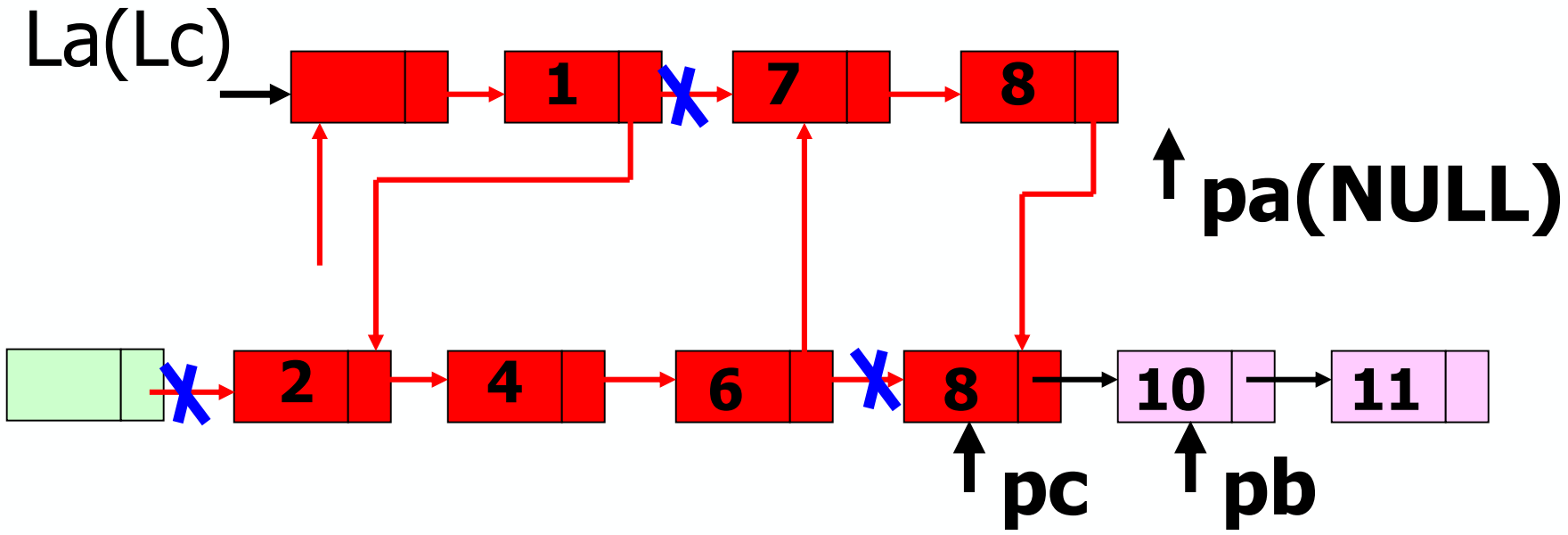

```
pc->next = pb;
pc= pb;
```

有序链表合并(pa->data >pb->data)



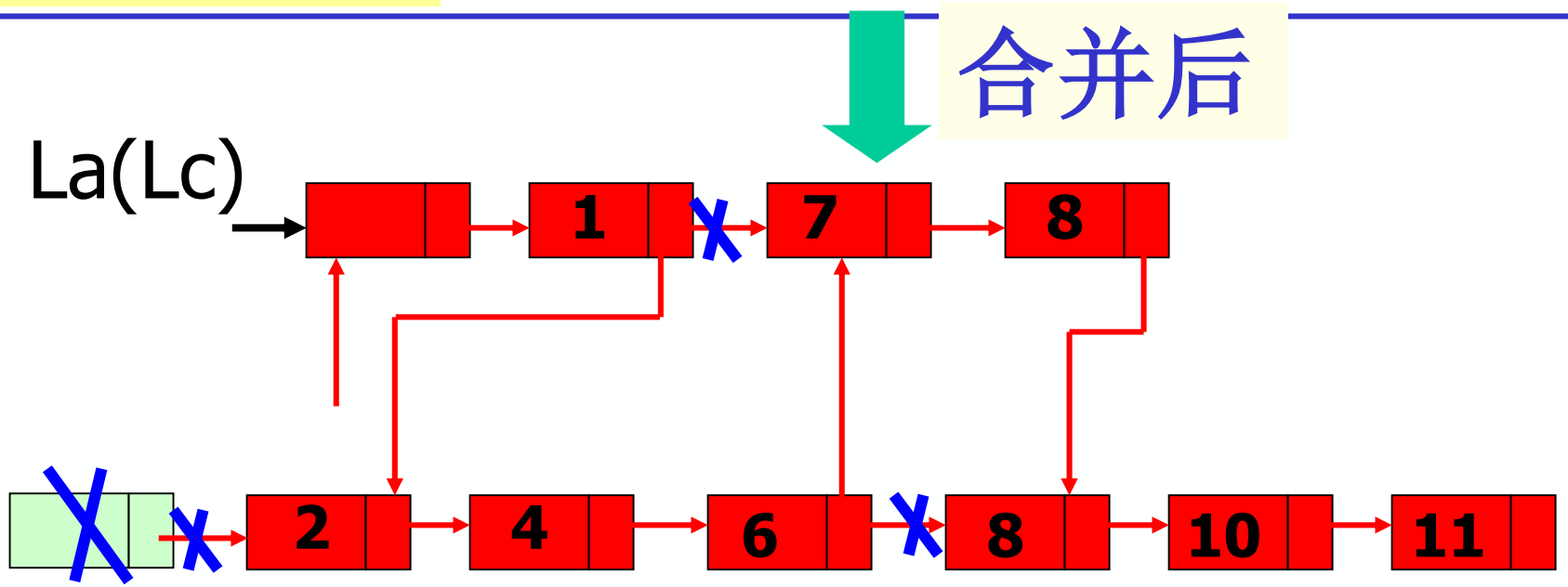

```
pc->next = pb;
pc= pb;
pb =pb->next;
```

有序链表合并




```
pc-> next=pa?pa:pb;
```

有序链表合并



delete Lb;

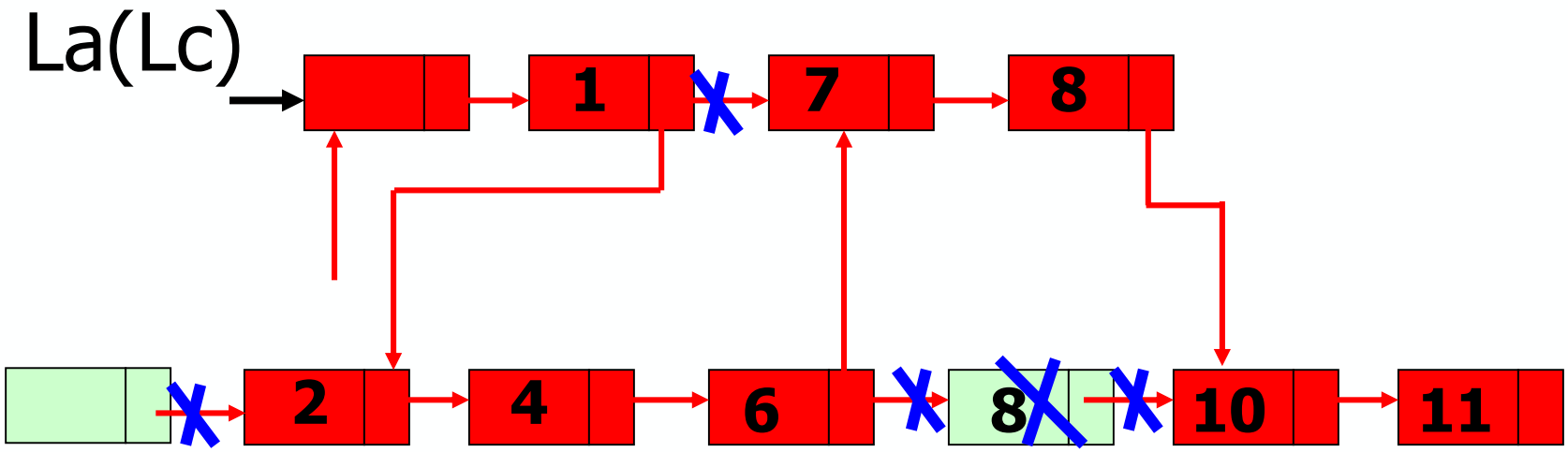
【算法描述】—有序的链表合并

```
void MergeList_L(LinkList &La,LinkList &Lb,LinkList &Lc){
  pa=La->next;  pb=Lb->next;
  pc=Lc=La;            //用La的头结点作为Lc的头结点
  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;    //插入剩余段
  delete Lb;           //释放Lb的头结点}
```

T(n)= $O(ListLength(LA)+ListLength(LB))$

S(n)= O(1)

思考1：要求合并后的表无重复数据，如何实现？



提示：要单独考虑

pa->data = =pb->data

思考2：将两个非递减的有序链表合并为一个非递增的有序链表，如何实现？

- ✓要求结果链表仍使用原来两个链表的存储空间，不另外占用其它的存储空间。

- ✓表中允许有重复的数据。

【算法步骤】

(1)Lc指向La

(2) 依次从 La 或 Lb 中“摘取”元素值较小的结点插入到 Lc 表的表头结点之后，直至其中一个表变空为止

(3) 继续将 La 或 Lb 其中一个表的剩余结点插入在 Lc 表的表头结点之后

(4) 释放 Lb 表的表头结点

2.8 案例分析与实现

案例2.1：一元多项式的运算

$$P(x) = 10 + 5x - 4x^2 + 3x^3 + 2x^4$$

数组表示

（每一项的指数i隐含在其系数p_i的序号中）

指数（下标i）	0	1	2	3	4
系数p[i]	**10**	**5**	**-4**	**3**	**2**

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

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

案例2.2：稀疏多项式的运算

多项式非零项的数组表示

（a）$A(x) = 7 + 3x + 9x^8 + 5x^{17}$

下标i	0	1	2	3
系数 a[i]	7	3	9	5
指数	0	1	8	17

（b）$B(x) = 8x + 22x^7 - 9x^8$

下标i	0	1	2
系数 b[i]	8	22	-9
指数	1	7	8

$$P_n(x) = p_1 x^{e_1} + p_2 x^{e_2} + \ldots + p_m x^{e_m}$$

线性表P =((p$_1$, e$_1$), (p$_2$, e$_2$),…,(p$_m$, e$_m$))

- 创建一个新数组c
- 分别从头遍历比较a和b的每一项
  - ✓指数相同，对应系数相加，若其和不为零，则在c中增加一个新项
  - ✓指数不相同，则将指数较小的项复制到c中
- 一个多项式已遍历完毕时，将另一个剩余项依次复制到c中即可

●顺序存储结构存在问题

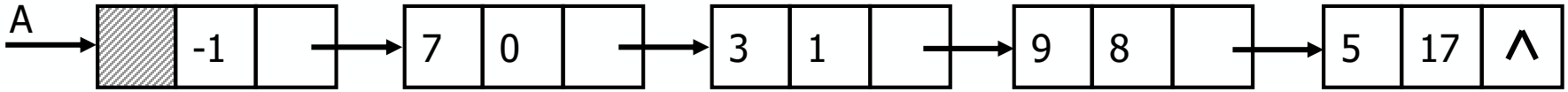
✓存储空间分配不灵活

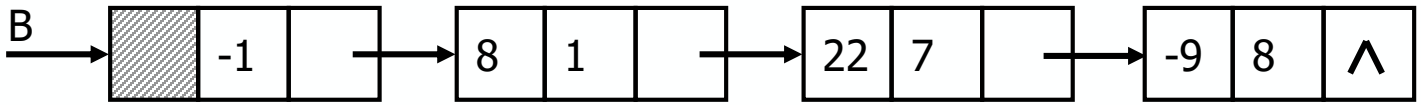
✓运算的空间复杂度高

链式存储结构

```
typedef struct PNode
{
  float  coef;//系数
  int   expn;        //指数
  struct PNode  *next;     //指针域
}PNode,*Polynomial;
```

$A_{17}(x)=7+3x+9x^8+5x^{17}$





$B_8(x)=8x+22x^7-9x^8$

多项式创建---【算法步骤】

① 创建一个只有头结点的空链表。

② 根据多项式的项的个数*n*，循环*n*次执行以下操作：

●生成一个新结点*s；

●输入多项式当前项的系数和指数赋给新结点*s的数据域；

●设置一前驱指针pre，用于指向待找到的第一个大于输入项指数的结点的前驱，pre初值指向头结点；

●指针q初始化，指向首元结点；

●循链向下逐个比较链表中当前结点与输入项指数，找到第一个大于输入项指数的结点*q；

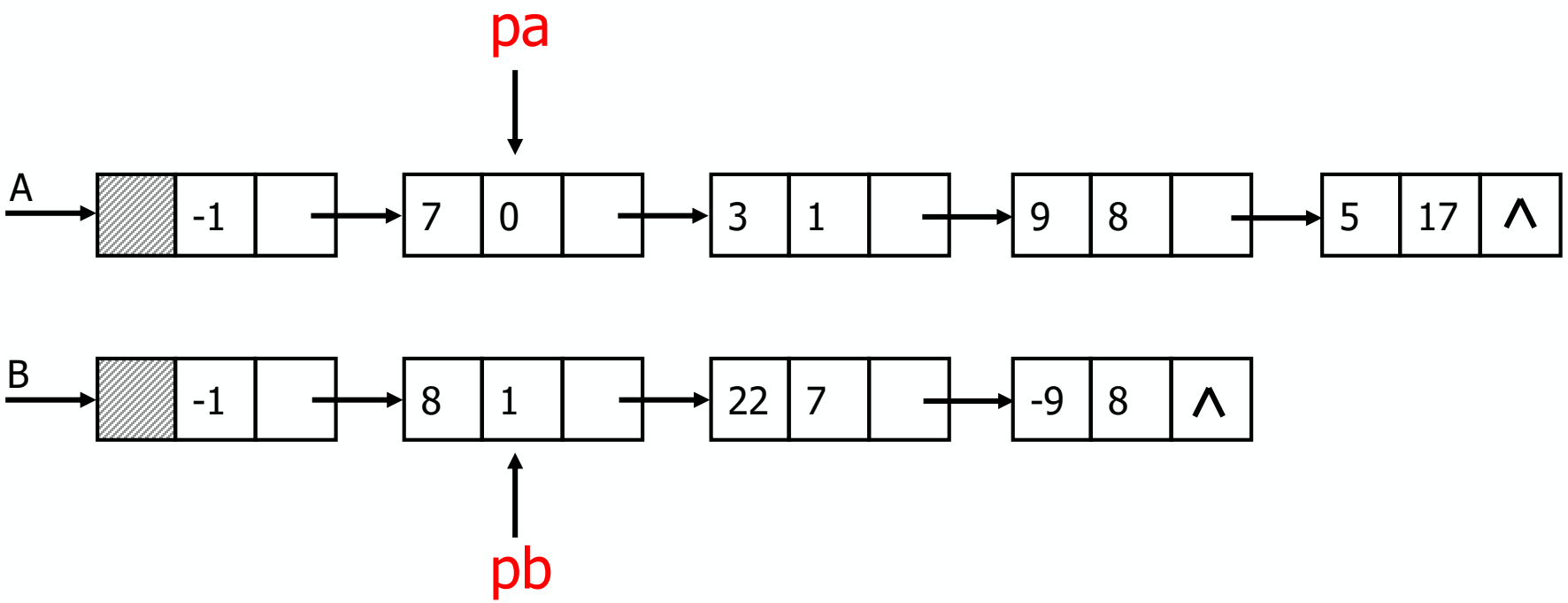
●将输入项结点*s插入到结点*q之前。

多项式创建---【算法描述】

```
void CreatePolyn(Polynomial &P,int n)
{//输入m项的系数和指数，建立表示多项式的有序链表P
  P=new PNode;
  P->next=NULL;                    //先建立一个带头结点的单链表
  for(i=1;i<=n;++i)                //依次输入n个非零项
  {
    s=new PNode;                   //生成新结点
    cin>>s->coef>>s->expn;         //输入系数和指数
    pre=P;                         //pre用于保存q的前驱，初值为头结点
    q=P->next;                     //q初始化，指向首元结点
    while(q&&q->expn<s->expn)      //找到第一个大于输入项指数的项*q
    {
      pre=q;
      q=q->next;
    }                              //while
    s->next=q;                     //将输入项s插入到q和其前驱结点pre之间
    pre->next=s;
  }                                //for
}
```

$A_{17}(x)=7+3x+9x^8+5x^{17}$

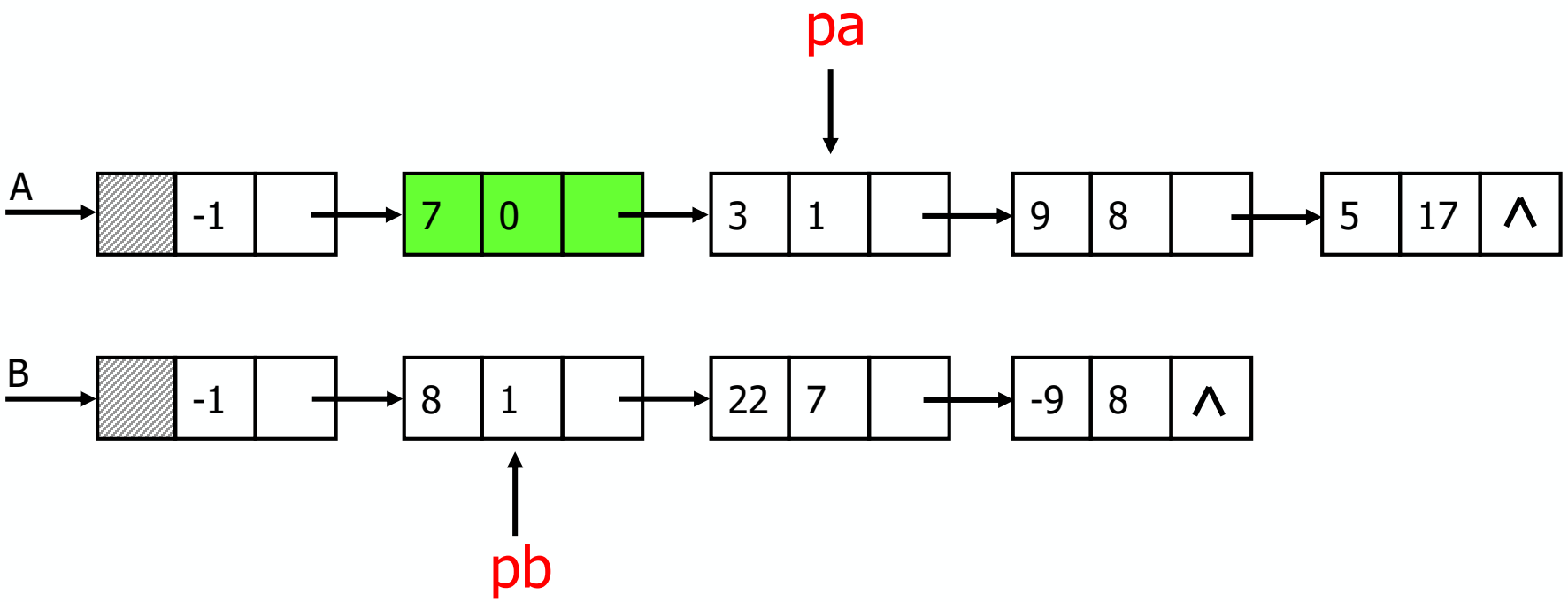
$B_8(x)=8x+22x^7-9x^8$



多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

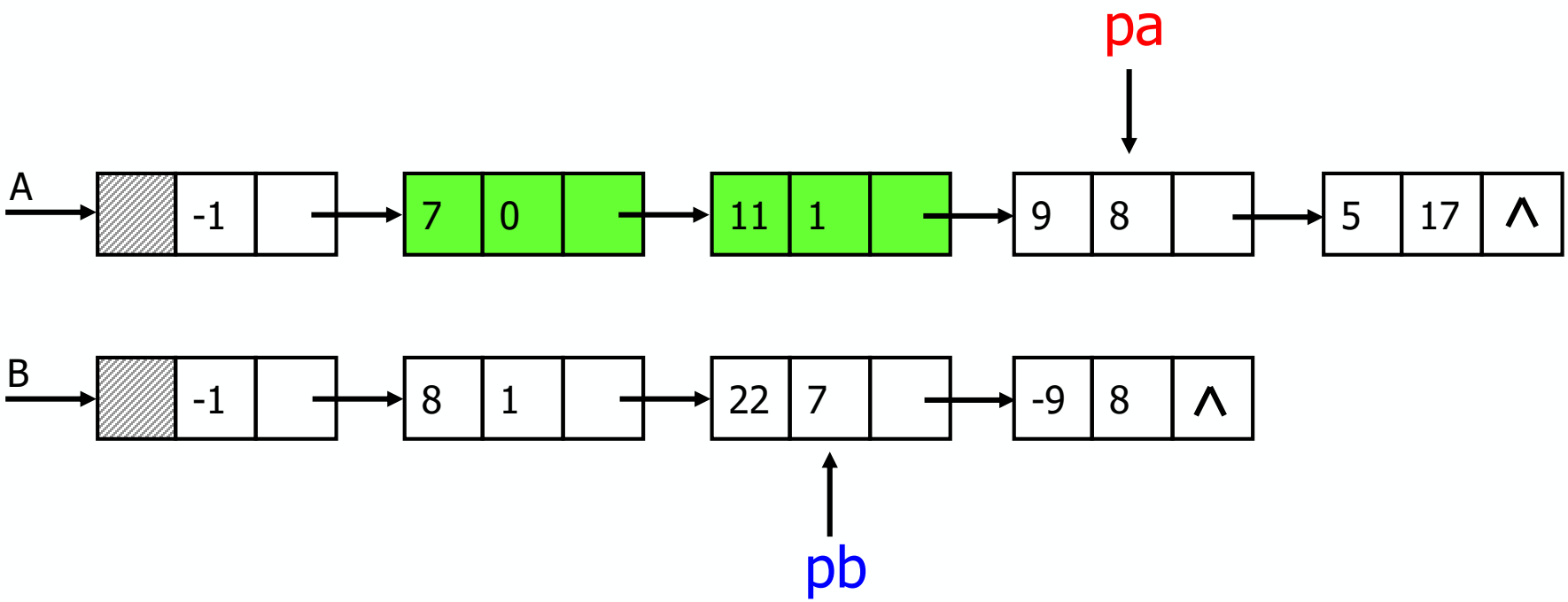
$B_8(x)=8x+22x^7-9x^8$



多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

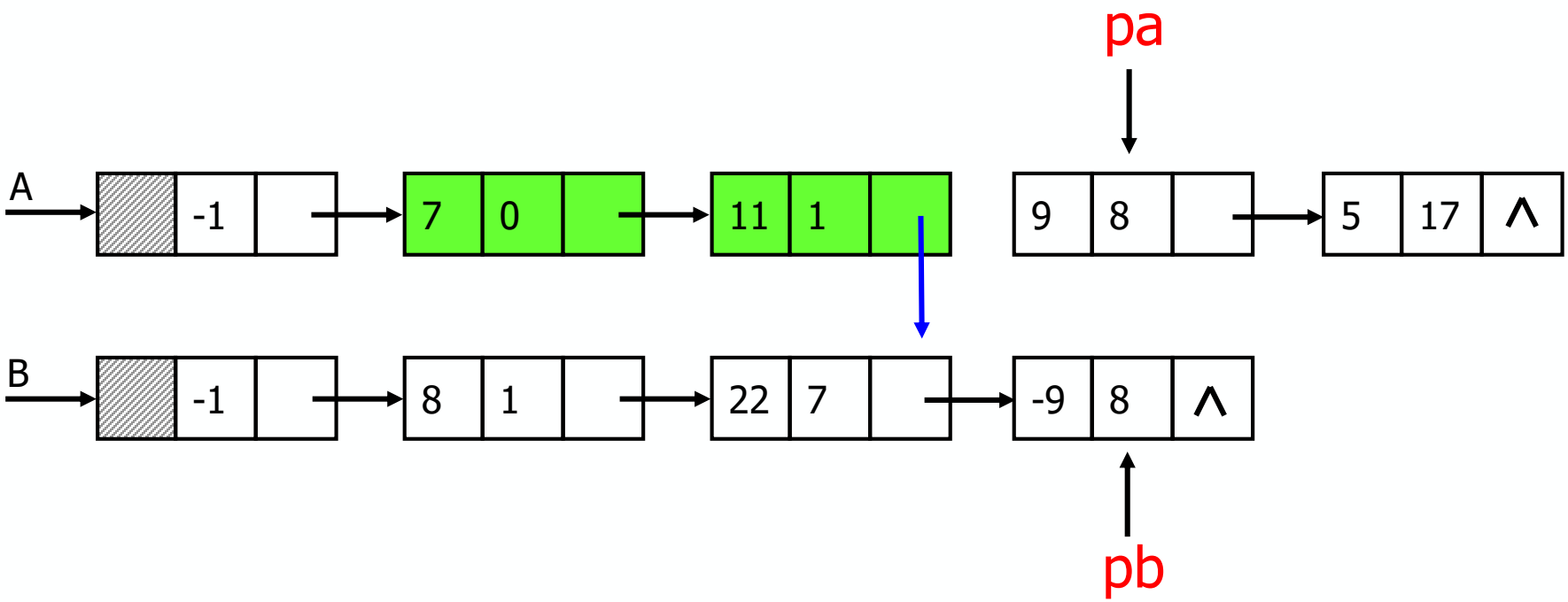
$B_8(x)=8x+22x^7-9x^8$



多项式相加

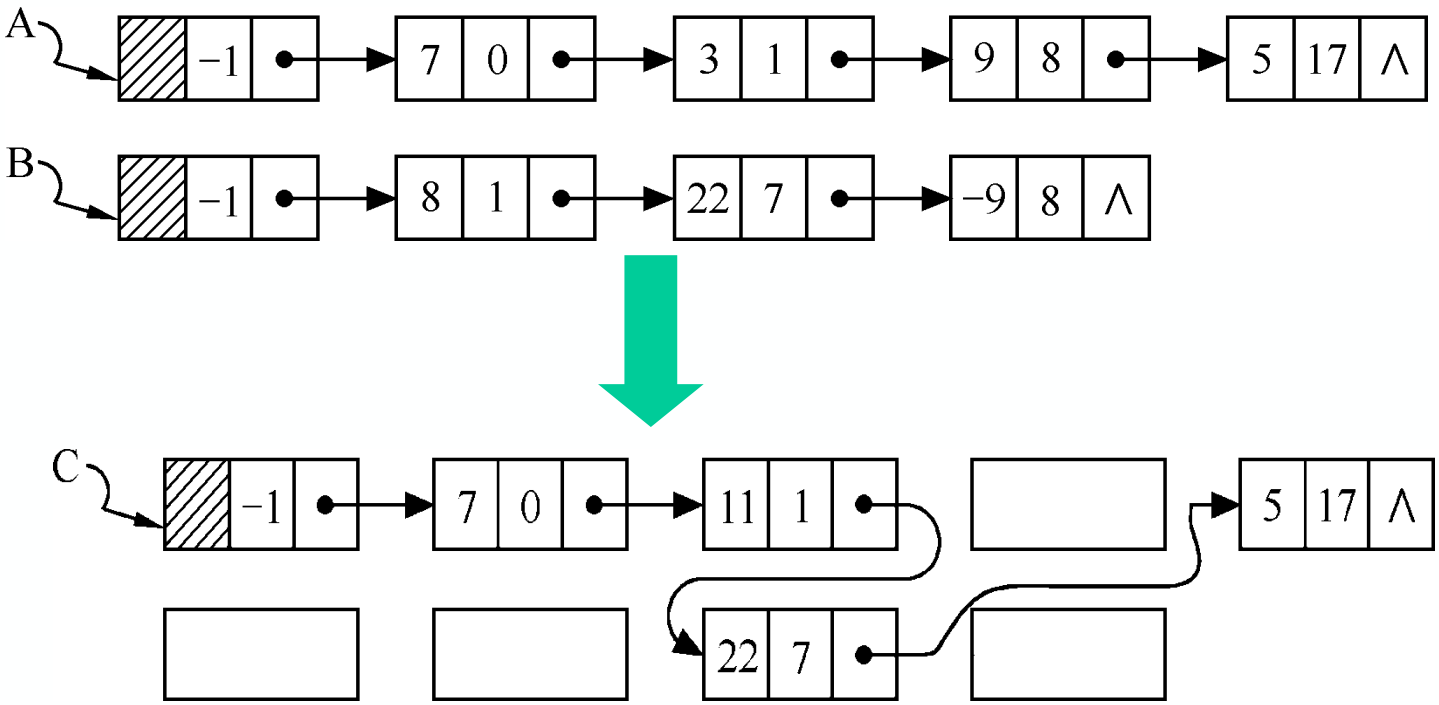
$A_{17}(x)=7+3x+9x^8+5x^{17}$

$B_8(x)=8x+22x^7-9x^8$



$A_{17}(x) = 7 + 3x + 9x^8 + 5x^{17}$

$B_8(x) = 8x + 22x^7 - 9x^8$



多项式相加---【算法步骤】

① 指针p1和p2初始化，分别指向Pa和Pb的首元结点。
② p3指向和多项式的当前结点，初值为Pa的头结点。
③ 当指针p1和p2均未到达相应表尾时，则循环比较p1和p2所指结点对应的指数值（p1->expn与p2->expn），有下列3种情况：

- 当p1->expn等于p2->expn时，则将两个结点中的系数相加，若和不为零，则修改p1所指结点的系数值，同时删除p2所指结点，若和为零，则删除p1和p2所指结点；
- 当p1->expn小于p2->expn时，则应摘取p1所指结点插入到“和多项式”链表中去；
- 当p1->expn大于p2->expn时，则应摘取p2所指结点插入到“和多项式”链表中去。

④ 将非空多项式的剩余段插入到p3所指结点之后。
⑤ 释放Pb的头结点。

案例2.3： 图书信息管理系统

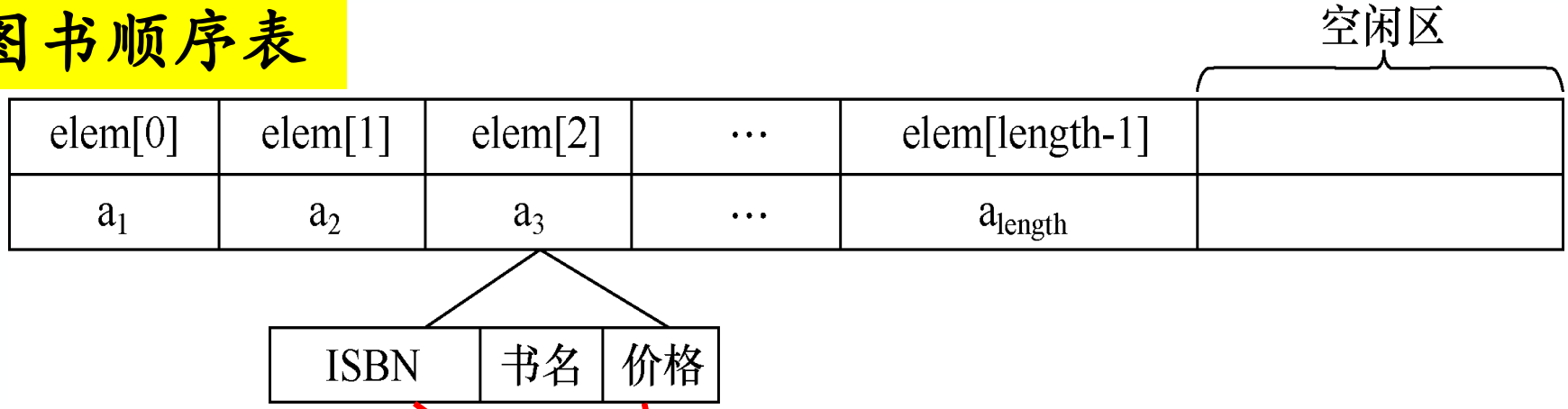
book.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

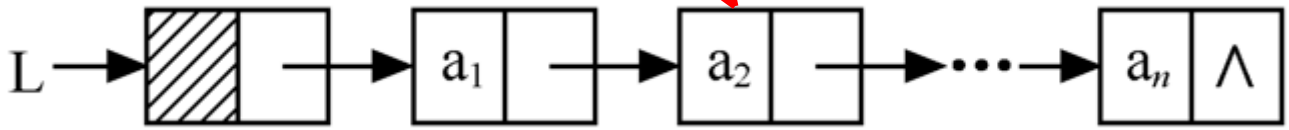
ISBN	书名	
9787302257646	程序设计基础	
9787302219972	单片机技术及应	
9787302203513	编译原理	
9787811234923	汇编语言程序设计教程	21
9787512100831	计算机操作系统	17
9787302265436	计算机导论实验指导	18
9787302180630	实用数据结构	29
9787302225065	数据结构（C语言版）	38
9787302171676	C#面向对象程序设计	39
9787302250692	C语言程序设计	42
9787302150664	数据库原理	35
9787302260806	Java编程与实践	56
9787302252887	Java程序设计与应用教程	39
9787302198505	嵌入式操作系统及编程	25
9787302169666	软件测试	24
9787811231557	Eclipse基础与应用	35

实验1---基于线性表的图书信息管理系统

图书顺序表


空闲区
elem[0]
elem[1]
elem[2]
…
elem[length-1]
a1
a2
a3
…
alength
ISBN
书名
价格

图书链表


L
a1
a2
an
∧

```
struct Book
{
  char id[20];//ISBN
  char name[50];//书名
  int price;//定价
};
```

```
typedef struct
{//顺序表
   Book *elem;
   int length;
} SqList;
```

```
typedef struct  LNode
{//链表
   Book  data;
   struct LNode  *next;
}LNode,*LinkList;
```

小结

1、掌握线性表的逻辑结构特性是数据元素之间存在着线性关系，在计算机中表示这种关系的两类不同的存储结构是顺序存储结构（顺序表）和链式存储结构（链表）。

2、熟练掌握这两类存储结构的描述方法，掌握链表中的头结点、头指针和首元结点的区别及循环链表、双向链表的特点等。

小结

3、熟练掌握顺序表的查找、插入和删除算法

4、熟练掌握链表的查找、插入和删除算法

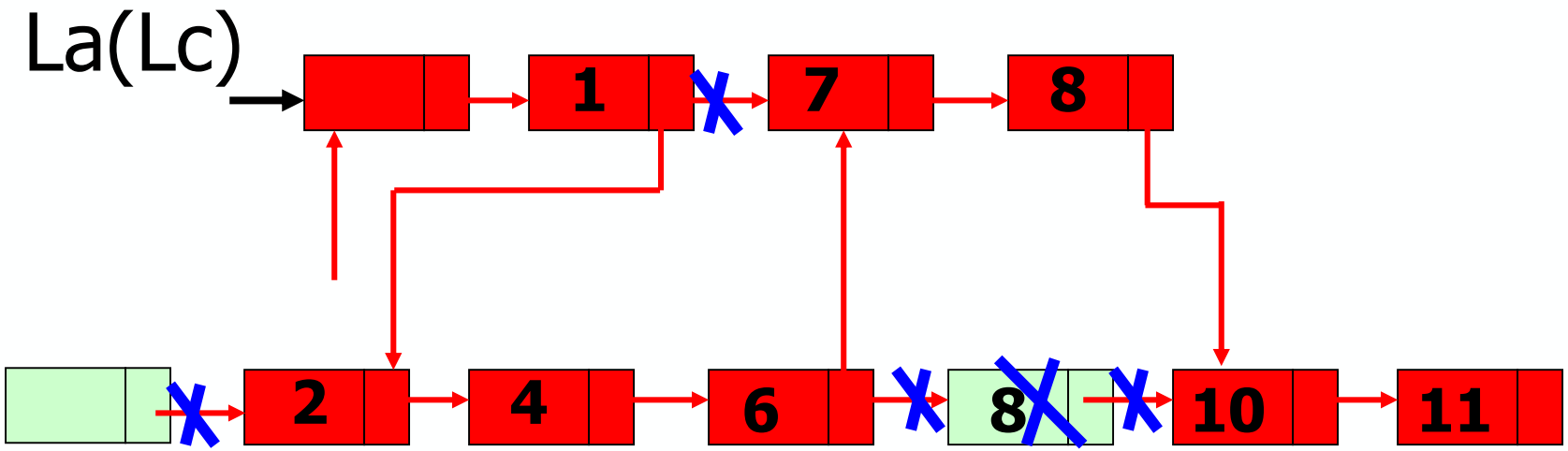
5、能够从时间和空间复杂度的角度比较两种存储结构的不同特点及其适用场合

		顺 序 表	链 表
空间	存储空间	预先分配，会导致空间闲置或溢出现象	动态分配，不会出现闲置或溢出现象
	存储密度	不用为表示结点间的逻辑关系而增加额外的存储开销，存储密度等于1	需要借助指针来体现元素间的逻辑关系，存储密度小于1
时间	存取元素	随机存取，时间复杂度为O(1)	顺序存取，时间复杂度为O(*n*)
	插入、删除	平均移动约表中一半元素，时间复杂度为O(*n*)	不需移动元素，确定插入、删除位置后，时间复杂度为O(1)
适用情况		① 表长变化不大，且能事先确定变化的范围 ② 很少进行插入或删除操作，经常按元素序号访问数据元素	① 长度变化较大 ② 频繁进行插入或删除操作

算法设计题

（1）将两个递增的有序链表合并为一个递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中不允许有重复的数据。

参考算法2.17



要单独考虑

pa->data = =pb->data

算法设计题

（2）将两个非递减的有序链表合并为一个非递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中允许有重复的数据。

【算法步骤】

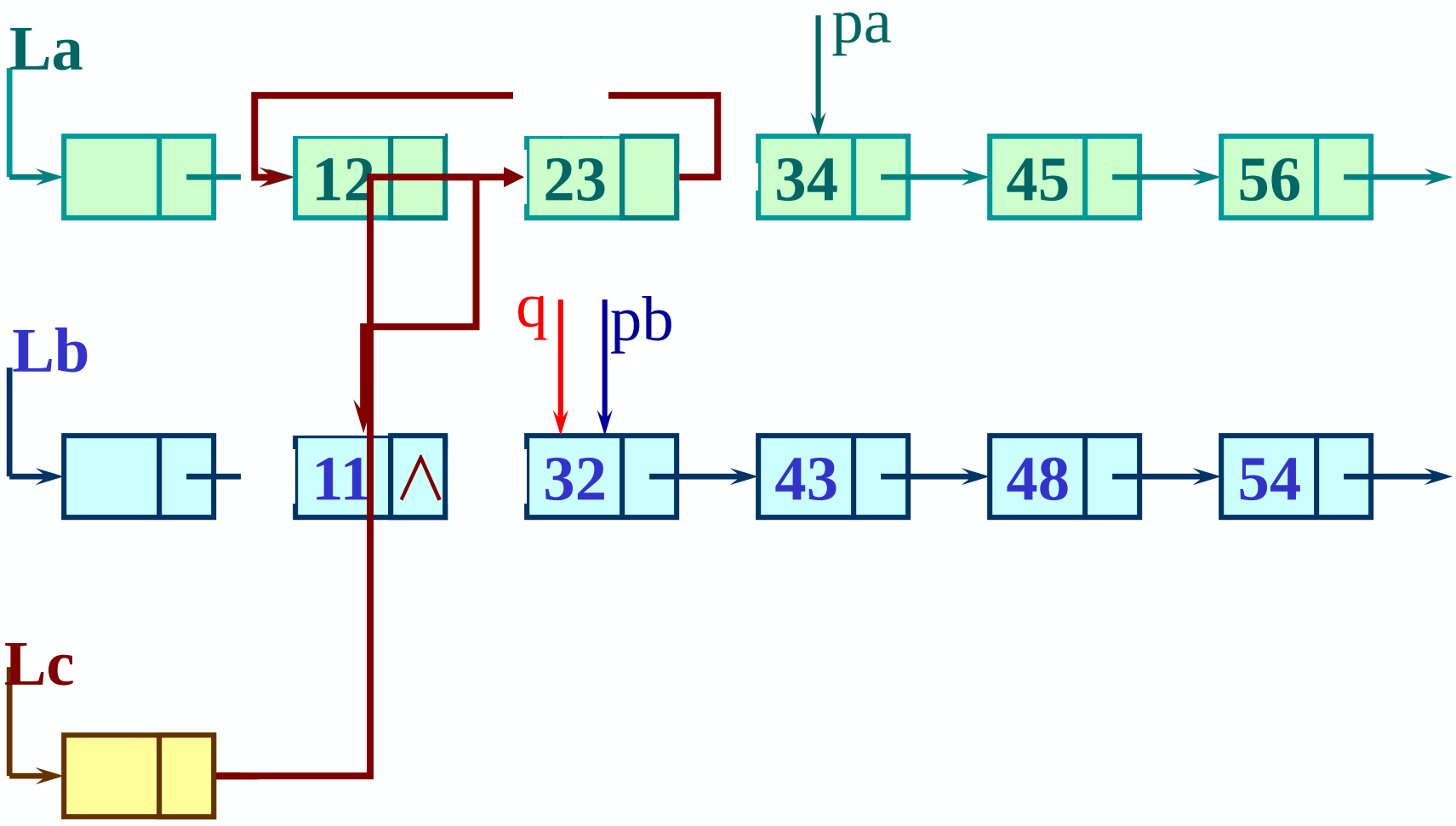
(1)Lc指向La

(2) 依次从 La 或 Lb 中“摘取”元素值较小的结点插入到 Lc 表的表头结点之后，直至其中一个表变空为止

(3) 继续将 La 或 Lb 其中一个表的剩余结点插入在 Lc 表的表头结点之后

(4) 释放 Lb 表的表头结点

第（2）题实现过程动态演示



算法设计题

（6）设计一个算法，通过一趟遍历在单链表中确定值最大的结点。

【算法步骤】类似于求n个数中的最大数

✓可假设第一个结点最大，用指针pmax指向。

✓然后用pmax依次和后面的结点进行比较，发现大者则用pmax指向该结点。

✓这样将链表从头到尾遍历一遍时，pmax所指向的结点就是最大者。

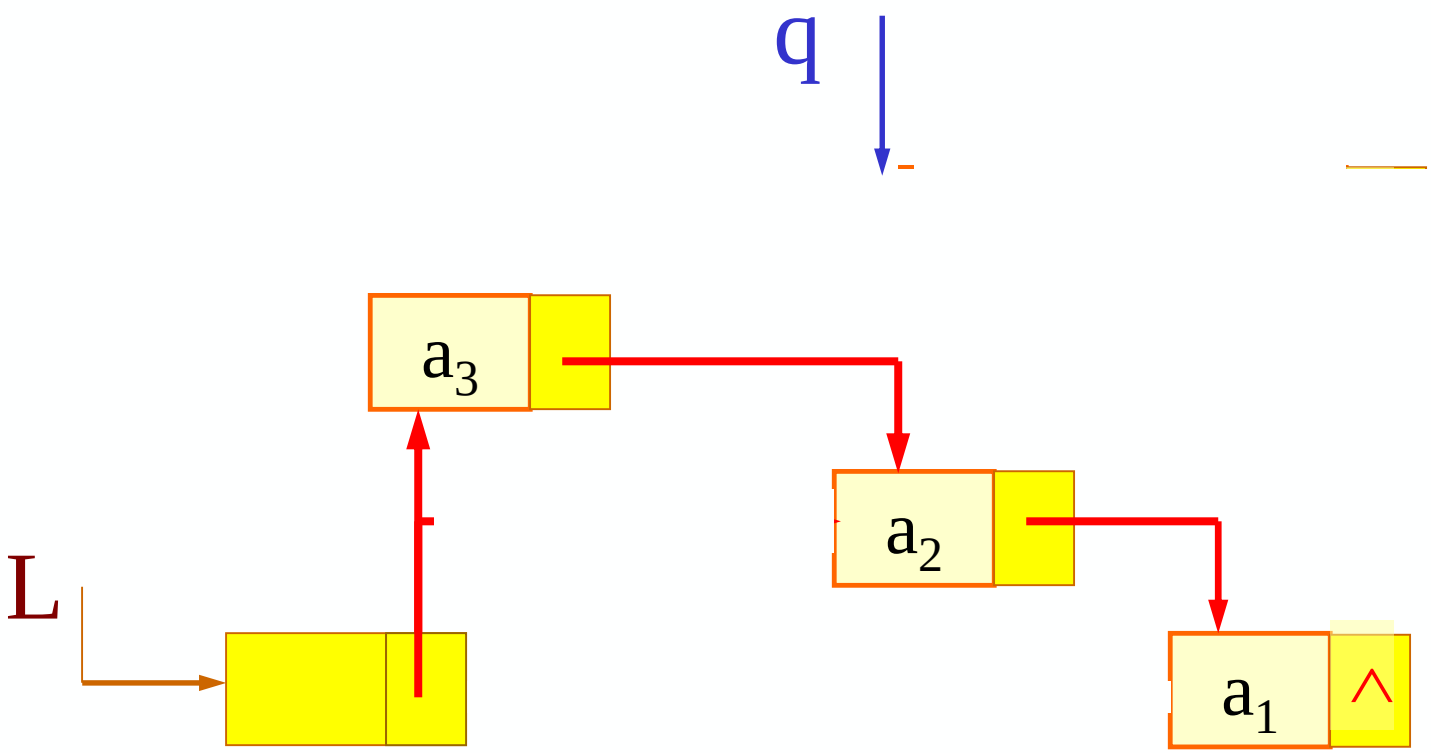
其中的比较语句形式如下：

```
if(p->data > pmax->data) pmax=p;
```

算法设计题

（7）设计一个算法，通过一趟遍历，将链表中所有结点的链接方向逆转，且仍利用原表的存储空间。

【算法步骤】从首元结点开始，逐个地把链表L的当前结点p插入新的链表头部



- ✓标志后继结点q
- ✓修改指针（将p插入在头结点之后）
- ✓重置结点p（p重新指向原表中后继）